23993

National Library of Canada | Bibliothèque nationale du Canada

CANADIAN THESES ON MICROFICHE | THÈSES CANADIENNES SUR MICROFICHE

NAME OF AUTHOR/*NOM DE L'AUTEUR* ANDREW BROWN

TITLE OF THESIS/*TITRE DE LA THÈSE* SIMULATION METHODOLOGY FOR COMPUTER SYSTEM MODELS

UNIVERSITY/*UNIVERSITÉ* ALBERTA

DEGREE FOR WHICH THESIS WAS PRESENTED/*GRADE POUR LEQUEL CETTE THÈSE FUT PRÉSENTÉE* MASTER OF SCIENCE

YEAR THIS DEGREE CONFERRED/*ANNÉE D'OBTENTION DE CE GRADE* 1975

NAME OF SUPERVISOR/*NOM DU DIRECTEUR DE THÈSE* DR. J. TARTAR

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

DATED/*DATÉ* Mar 14th 1975 SIGNED/*SIGNÉ* Andrew Brown

PERMANENT ADDRESS/*RÉSIDENCE FIXE* 9316 86 STR.
EDMONTON ALBERTA
T6C 3E7

NL-91 (3-74)

THE UNIVERSITY OF ALBERTA

SIMULATION METHODOLOGY FOR COMPUTER SYSTEM MODELS

by

© ANDREW McGREGOR BROWN

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

DEPARTMENT: COMPUTING SCIENCE

EDMONTON, ALBERTA

SPRING, 1975

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled SIMULATION METHODOLOGY FOR COMPUTER SYSTEM MODELS

submitted by Andrew McGregor Brown

in partial fulfilment of the requirements for the degree of Master of Science.

John Tartar
Supervisor

Keith A. Browne

W.S. Adam

Date Feb. 6 1975

DEDICATION

To my late father, Charles Frank Brown

ABSTRACT

This thesis deals with the use of simulation in the evaluation of the performance of computer systems. The scope of the thesis is restricted to the study of digital procedural models and the objective is to develop a methodology for simulating computer systems. This methodology defines a step by step procedure for building a procedural model of a computer system. A brief historical perspective is presented to familiarize the reader with current problems and methods of solution. Procedural simulators are analyzed and a fundamental set of structural characteristics and operational characteristics are defined. These characteristics are then used to classify simulation models. A methodology is developed for constructing procedural models. It consists of a notation for describing a job mix and hardware characteristics, a structure for defining an operating system and it states measures which are used for determining system performance. The methodology allows for the simulation of various sections at different levels of detail. The implementation of the methodology is described and validation techniques are discussed. A formal validation technique for this methodology is proposed. Two applications of the methodology are outlined to demonstrate the versatility of the methodology. Finally, the results are summarized, some limitations are indicated, and some areas for possible future study are suggested.

ACKNOWLEDGEMENT

I would like to acknowledge the assistance of the University of Alberta, Department of Computing Science, for its financial support during the course of this research. I would like to thank Dr. Tartar for his continued encouragement and guidance. In addition, I would like to express my appreciation to my wife, Linda, for her patience and her invaluable assistance in typing the manuscript.

TABLE OF CONTENTS

CHAPTER		PAGE
1	INTRODUCTION	1
	1.1 Background and Objectives	1
	1.1.1 Thesis Outline	1
	1.1.2 Basic Simulation	2
	1.1.3 Reasons for Simulation	6
	1.1.4 Limitations of Simulation	6
	1.1.5 Justification/Motivation	7
	1.2 Early Developments	8
	1.2.1 Outline of Early Problems	8
	1.2.2 Classification of Major Problems	10
	1.2.3 Early Simulation of Computer Systems	12
	1.3 Neilsen Simulator	14
2	ANALYSIS OF MODEL STRUCTURE AND CHARACTERISTICS	18
	2.1 Fundamental Concepts	18
	2.2 Analysis of Model Structure	22
	2.3 Event Oriented Models	23
	2.3.1 Neilsen's Simulator	23
	2.3.2 McDougall's Simulator	23
	2.4 Trace Driven Models	24
	2.4.1 Cheng's Trace Driven Simulation	24
	2.4.2 Noe and Nutt's Trace Driven Simulation	25
	2.5 Generalized Models	26

TABLE OF CONTENTS (continued)

CHAPTER		PAGE
3	CLASSIFICATION OF MODELS	29
	3.1 Basic Model Structure	29
	3.2 Identification of Operational Characteristics	31
	3.3 Classification of Models According to Characteristics	33
4	DESCRIPTION OF MODEL METHODOLOGY	37
	4.1 Introduction	37
	4.2 Job Generation	38
	4.3 Hardware Configuration	45
	4.4 Operating System	48
	4.4.1 Job Initiator and Control	50
	4.4.2 Memory Management Scheme	50
	4.4.3 Scheduling Routine	51
	4.4.4 Input/Output Processor	52
	4.5 System Performance	53
5	IMPLEMENTATION AND VALIDATION	58
	5.1 Implementation	58
	5.2 Establishing a Need for Validation	62
	5.3 Types of Validation Procedures	63
	5.4 Formal Validation Procedure	65
6	APPLICATION OF THE METHODOLOGY	68
	6.1 Simulation of a Uniprogrammed System	68
	6.2 Simulation of a Multiprogrammed System	73

TABLE OF CONTENTS (continued)

CHAPTER		PAGE
7	CONCLUSIONS	77
	7.1 Research Results	77
	7.2 Limitations of the Methodology	79
	7.3 Future Areas of Study	80
BIBLIOGRAPHY		82

LIST OF TABLES

TABLE		PAGE
1	HARDWARE CHARACTERISTICS	47

LIST OF FIGURES

FIGURE		PAGE
1	SIMULATION REQUIREMENTS	5
2	COMPUTER MODELLING AND SIMULATION	20
3	COMPUTER SIMULATION MODELS	34
4	MODEL GENEALOGY	35
5	MODEL METHODOLOGY	37
6	RESOURCES	54
7	IMPLEMENTATION	59
8	IMPLEMENTATION (continued)	60
9	UNIPROGRAMMED OPERATING SYSTEM	72
10	MULTIPROGRAMMED OPERATING SYSTEM	75

CHAPTER 1

INTRODUCTION

1.1 Background and Objectives

1.1.1 Thesis Outline

Simulation has recently come to the forefront as a major tool for the analysis of the performance of computer systems. This thesis deals with the use of simulation to study the performance of computer systems. In particular it outlines a method which can be used to simulate computer systems. The method defines the steps which are necessary for the construction of a procedural simulator. Four basic tasks must be completed. The job mix must be established. A hardware configuration must be defined. An operating system must be specified and measures of system performance must be included. The methodology provides the tools with which an analyst can outline each of the above, and it shows the sequence of events which must take place in order to implement a procedural simulator. Specifically, in Chapter 1, some background material will be presented to familiarize the reader with the problem under study. Chapter 2 analyzes procedural models of computer systems to determine model characteristics and in Chapter 3 these characteristics are used to classify models according to their structure. A complete description of the model methodology is presented in Chapter 4. In Chapter 5 the implementation and validation of the methodology is discussed. For illustrative purposes two

applications are outlined in Chapter 6. It is not intended that the two examples be compared. Finally, in Cahpter 7 the results are summarized and presented along with a discussion of future trends and areas for exploration.

Section 1.1 states the objectives and scope for this study and provides a background to simulation. The advantages and disadvantages of simulation are discussed. In addition, this section provides some justification and motivation for the study of these particular problems.

Performance evaluation of systems in general is a very wide field and to attempt to develop a methodology to cover all situations would be a formidable problem. As a result, this thesis is limited to a discussion of simulation of computer systems. Procedural models are considered as the vehicle for the development of the methodology since this type of model does provide the necessary flexibility. The ideal parameterized program capable of simulating any system is perhaps not yet feasible. This methodology is not intended to be the ideal parameterized simulator. However, it is intended for those users who do wish to simulate a particular system. Models of parts of a computer system can be organized by an analyst and then, using the guidelines set forth in the methodology, the whole computer system can be simulated. Throughout the thesis a wide range of problems are considered so that the resulting methodology can have general applicability. The objective is to develop a methodology which can be used to set up procedural models of computer systems.

1.1.2 Basic Simulation

This section outlines the fundamentals of simulation and

gives examples of the main objectives for a simulation study. Some of the advantages and disadvantages are stated, including a brief description of the uses of simulation. The development of simulation is traced from the early periods of computers to the point where several complex procedural models of computer systems existed.

The main objectives for any simulation study are usually to determine some feature of the system, for example, its behaviour or the capacity of the system. Simulation can be used to make a feasibility study, to assist in procurement decision-making, for design support, to improve performance or to determine capacity. In general simulation is used as an experimental tool or as an evaluation and performance guide. It also has use in learning about new systems, as a teaching tool, and as a means of projecting future requirements. There are many areas in which simulation has been used with beneficial results.

A brief review of the basic steps of a computer simulation study is appropriate at this point.

1. Definition of the Problem. This includes a definition of the general objectives of the study, a detailed specification of the questions to be answered and a description of the means to be used to achieve the desired results.
2. Planning the Study. Once a definite problem has been established and a line of attack decided upon, an analyst can schedule the remaining steps of the study according to time, manpower, financial and other restrictions.
3. Formulation of a Model. The first step in the development of the model is to formulate a model by collecting

and analyzing data pertaining to the system, identifying major system elements and defining system interactions.

4. Construction of a Computer Program for the Model. Take the conceptually valid model that has been produced and flow chart, code, and debug the model.
5. Validation of the Model. Compare results of several initial simulator runs with hypothetical, actual and/or historic data. This is probably the most critical step in the formulation of the model.
6. Design of Experiments. Evaluate alternative experimental design techniques, select one and define, in terms of the technique to be used, the experiments to be performed.
7. Execution of Simulation Runs. Prepare or select appropriate input data and run the simulator using this data.
8. Analysis of Results. Investigate results and compare them to the objectives or expected results from Step 1. Cycle through Steps 6, 7, and 8 until the desired objectives have been met or other considerations halt the experimental process.

SIMULATION REQUIREMENTS

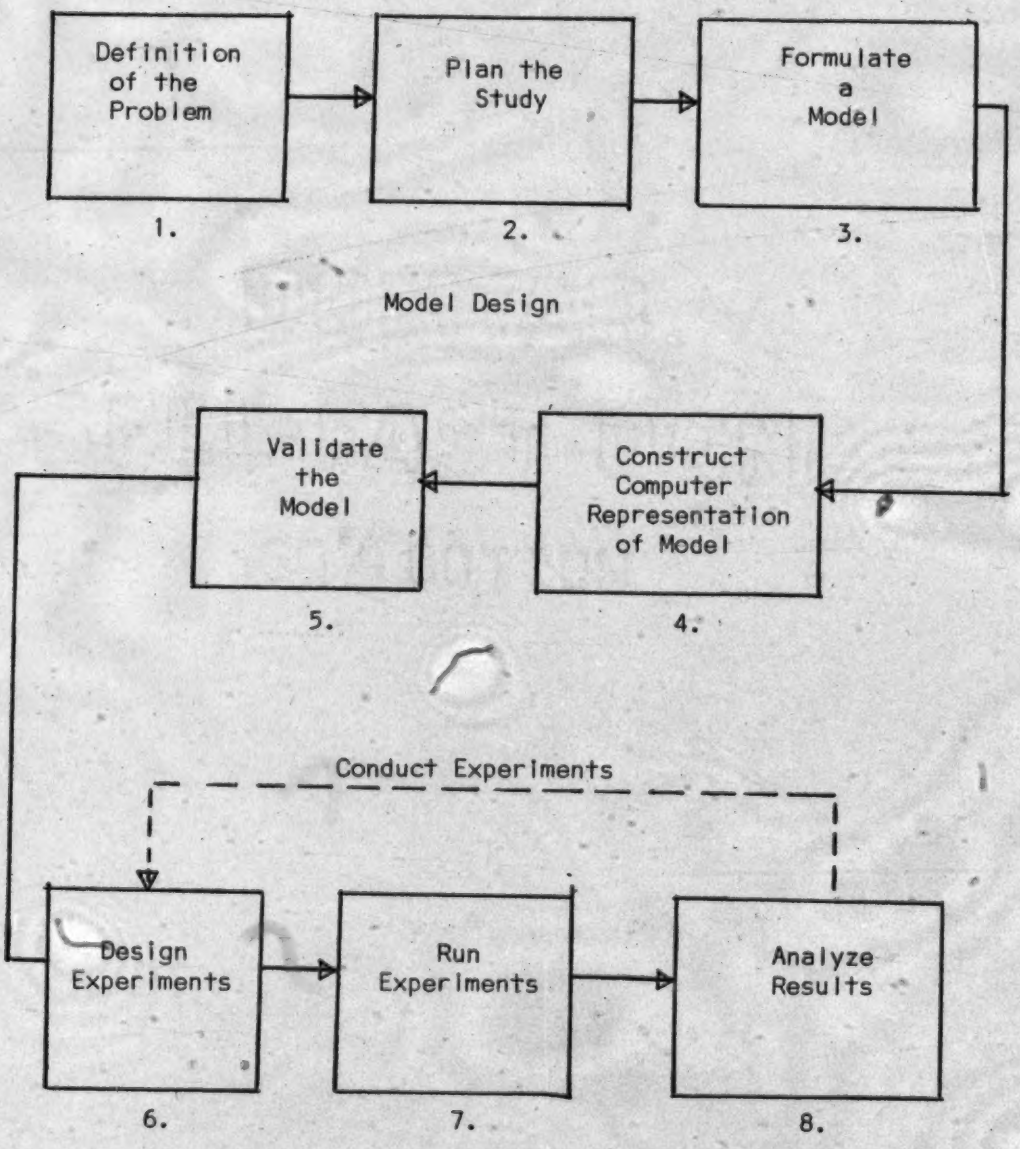


Figure 1

1.1.3 Reasons for Simulation

There are many reasons why simulation is used. Among the most common are the ones that cover the areas of procurement decision-making, analysis of system performance and extrapolation of results for prediction purposes. Processes that require a long time to complete in the real world can be simulated and run on a computer in a short time. Simulations may not be restricted by real world constraints on real systems. Chance elements can be monitored and controlled and models can be designed so that they are flexible and adaptable. Some very complex systems defy analysis by other methods.

Purely mathematical models may be extremely complex. Experimentation with pilot systems is often very costly. Relevant variables are not always under control. Intuition and experience can be very inadequate. Therefore, whenever one or more of these conditions is true, simulation becomes a valuable tool. Chorafas [9] says that no other formal method can provide the same type of results than can be obtained from simulation.

Simulation may be a technique of last resort and yet, much effort is now devoted to develop computer simulation because it is a technique that gives answers, in spite of its costs and time required for development.

1.1.4 Limitations of Simulation

Simulation, though, is not without problems. Although it is a very well known technique, most people are not familiar with <u>how</u> to use it and as a result it is not a widely accepted technique. Some people believe the approach is awkward to use. Results are not always

trusted and a simulation study can take a considerable amount of time to complete if it is poorly organized.

Some of the more common problems of implementation are the great effort required in programming. Nowadays special simulation languages are available that ease this burden somewhat if an analyst is familiar with the languages. A very adequate representation of the real world at the proper level of detail and with the proper accuracy is required. Correct validation of a model is the most important limitation. No useful conclusion can be drawn from an improperly validated model.

Simulation is an investigative technique. That is, it is not used to produce optimal solutions or mathematically "best" solutions out of all possible solutions. The choice of solution is generally a function of management. This choice is generally between one or two that are judged to be the best of several tried. Only by very careful planning can the pitfalls be avoided and some useful results and information obtained from a simulation.

1.1.5 Justification/Motivation

In the review of literature on computer simulation models, it became apparent that many approaches to computer model building had been tried. As a result many different types of models have been built and a great deal of useful information has been forthcoming. Each individual model could be said to be satisfactory (at least those that worked), however when considered as a group, they lack a general cohesiveness.

Several recent papers, especially the one by De Cegama [12], have stressed the idea of a computer model building methodology or more simply an overall approach to model building and not just an out of context look at a part of a system.

This thesis formalizes the computer model building explosion by analyzing model structure, classifying these models into an order and identifying desirable characteristics. It presents a methodology for building procedural models of computer systems that will have general application. Hopefully it will reduce the cost and time involved in developing specific models. Perhaps by approaching simulation of computer systems in this manner, implementation can be simplified and performance of systems can be improved.

This section has provided a general outline of the content of the thesis. It has introduced simulation and its advantages and disadvantages. The scope of the study was outlined and the objectives for the research were stated. The justification and motivation for this undertaking were also presented. The next section provides an insight into early problems of computer systems and early simulations of computer systems.

1.2 Early Developments

1.2.1 Outline of Early Problems

In the beginning, simulation had not developed to a point where it was being applied to analyze the performance of computer systems themselves. By the early sixties, however, analysts became concerned with numerous problems concerning overall computer system performance. Overlay problems were studied. Methods of increasing performance by

overlapping processing with input/output operation were developed. Strategies were developed which attempted to maximize throughput or minimize turnaround time.

By the mid sixties the concepts of multiprogramming, multiprocessing, and time sharing came into more prominent use. These concepts brought with them a set of problems that were an order of magnitude greater in complexity. That is, individual parts of systems were being rigorously studied and optimized. Computer operating systems were built which paid attention to the production of efficient object code, conversion problems including emulation, and multiplexing slow peripherals.

Simulation was applied to study these and similar problems and by the late sixties extensive simulation projects had been carried out in such areas as:

1. comparison of off-line and on-line programming performance,
2. time sharing,
3. user console behaviour,
4. definition of input job mix characteristics,
5. disk arm contention,
6. deadlock,
7. thrashing,
8. resource management including scheduling algorithms,
9. memory protection, and
10. the entire area of virtual memory systems.

In the last few years particular attention has been paid to the emerging computer networks. Simulation programs are now being

developed to study the major performance aspects of computer networks. Studies of the use of minicomputers, message switching, front end communications processors, and fast memory caches are all yielding important information. The next section will outline the major areas of study within computer systems.

1.2.2 Classification of Major Problems

Many of the problems of the late sixties are still with us. No satisfactory solution has been obtained to some problems although a great deal of insight into these problems has been gained. Denning [13] divides the problems of operating systems into five broad major areas. These are:

1. Programming
2. Storage Allocation
3. Concurrent Processes
4. Resource Allocation
5. Protection

In the area of programming, Denning considers four sub-problems. Programs should be modular for ease of programming, maintenance, and documentation. If at all possible, programs should be machine independent to allow for transportability. Programs should have the capability for defining and processing structured data. Finally, there should be greater use of high level languages.

With regard to storage allocation, the problems can be divided into those that apply to computational store, and those that apply to long term store. Virtual memory concepts emerged as the solution to overcome poor storage allocation by the user. The virtual

memory concept also satisfies the requirements of multiprogramming, relocation, and protection. Virtual memories, however, are not suitable for permanent storage because the dynamics of information additions and deletions require some method of managing names. Two methods that are being used to achieve long term store are file systems and segmented name space systems.

The problem of concurrent processes deals with how to synchronize and separate processes that are running concurrently on the computer system. The deadlock problem, where several concurrent processes can tie up a system although each individual process does not demand all the resources of the system, is included in this category.

Resource allocation has been defined as the need to regulate resource usage by processes in order to optimize system efficiency and provide good service to all users. Long term pricing policies can be used to damp peak demand, but short term policies are necessary to alleviate the burden of resource allocation from the programmer, to control the interference and interaction among processes which result from sharing, to control multitasking which exposes the system to deadlock, and to regulate the competition for resources.

Protection is a three level problem. Techniques must exist which will work within the system to control accesses by processes to other system processes. Access must be denied to confidential data. Some protection must exist against system failure.

These five major areas define the current problem areas for computer systems. These are by no means all of the problems but they do indicate the direction of major investigations today.

1.2.3 Early Simulation of Computer Systems

Computer simulation has been used since the emergence of computers. Instead of a purely mathematical approach we can easily program a computer to follow a set of instructions that mimic a real life situation. At first, applications were restricted to small simple models. As computers increased in power it became feasible to consider the simulation of larger systems. Simulations of commercial, scientific, and military systems have all been accomplished since the advent of the computer. A natural extension took place when it became clear that computer systems themselves were becoming complex. This extension led to the early simulations of computer systems.

Early simulations of computer systems involved first and second generation machines. Since most of these machines have been replaced by third generation machines (and third generation problems), it is more interesting to begin to look at the efforts made in simulating computer systems at the point when these machines were introduced. This was approximately the mid sixties.

One of the earliest studies of simulations of computer systems, in particular a time sharing system, was that conducted by Scherr [50] in 1966. He attempted to develop techniques and models for the analysis of a broad class of interactive, time shared computer systems. Simulation models were used to study a system similar to Project Mac's Compatible Time Sharing System (CTSS) and continuous Markov processes were used to examine more general classes of time shared systems. His results showed that it was possible to model accurately, users of interactive computer systems and the systems themselves by means of relatively simple models. Katz [23, 24] in 1966 and

1967 and Fine and McIsaac [18] in 1966 also produced noteworthy results.

Then in 1967 Neilsen [41,42] at Stanford constructed a simulator for the IBM 360/67. Briefly Neilsen selected as his objective the solutions of questions concerning the proper configuration of systems. His approach was to build a model of a general purpose time sharing system which included highly parameterized input for specifying hardware characteristics and isolated sections of code for describing software algorithms. The resulting model was very flexible and could be altered to reflect any hardware/software configuration. The problem of configuring the IBM 360/67 time sharing system for the Stanford University Computation Center was selected as an illustrative problem and extremely useful results were obtained.

Unfortunately, some simulation models contain too much detail. This results in such complexity that one minute of simulated time can take an inordinate amount of real central processing unit (CPU) time.

Another study of a CDC 6000 series computer was conducted by McDougall [35] in 1967. In 1969 Pinkerton [45] employed a software monitor on an IBM 360/67, operating under the Michigan Terminal System (MTS), to monitor operating system and user program behaviour and performance. This paper illustrates an alternative to simulation since by no means is simulation the only method of determining and controlling system performance.

Seaman and Soucy [52] in 1969 also completed a study of computer systems using simulation. They aimed at developing a model which would serve both installation planners and program developers. The simulated system consisted of three major components. These were a configuration base, an operating system submodel and applications

programs. Useful models were developed although conclusions indicated that the two sets of users mentioned above were incompatible and more realistically models should be developed with the appropriate level of detail for each type of user.

During the early seventies there was a great proliferation of models that simulated increasingly complex and detailed substructures. There are too many of them to mention here but illustrative papers are those by Noe and Nutt [44]; Dahle and Pienne [11]; Bowdon, Mamrak and Saltz [6]; and Rourke et al [48].

Some of the early problems of simulation have been examined and some of the more recent problems have been classified. In addition, the development of simulation, as related to the simulation of computer systems, has been presented to acquaint the reader with the forms of attack on problems in this area. The last major section of this chapter discusses an existing simulation model that will later be used as a basis for developing the methodology.

1.3 Neilsen Simulator

Because a great deal of the work involved with this thesis revolves around the simulator of an IBM 360/67 originally written by N. R. Neilsen, it is necessary to describe that simulator in some detail. The reader will then be in a position to more fully appreciate the development of the methodology. This section presents an outline of Neilsen's simulator.

The basic model had to meet several fundamental requirements. It had to provide the capability to specify a system, to modify key algorithms, and to determine the effect of various job mixes. These

requirements had implications with regard to the level of detail of the model. It had to be sufficiently detailed to model a paging environment since the model was to be general in nature. The model had to keep track of every page and job in the system. The basic unit of time and storage were chosen to be 100 microseconds and 1 page.

Actual jobs were represented by deterministically setting up a sequence of page references and supervisor requests appropriately spaced by execution times and then linking and/or repeating several sequences to represent an entire program. Consequently the sequences for a particular job were constructed from a set of master sequences which represented a prototype for each different job type.

A job description language was developed for the specification of sequences. Construction of prototypes was a "once only" task, and selection of job mix then became a simple task of specifying the probability of a particular type of job being initiated.

It was necessary to build a serial model of a parallel process to represent the multiprocessor configuration. As a result simulated events were designed as independent activities and each CPU had its own clock as well as the master clock. Software algorithms were isolated for easy replacement or adjustment.

To assess the performance of the system, statistics were gathered with respect to the user (response time by priority class or job type), with respect to the CPU (utilization including overhead and idle time), and with respect to the I/O devices (items such as queue sizes, average waiting time for read and write to and from devices).

Page accounting within the model was accomplished by using arrays to keep track of each page in each of the various categories of pages. For example, one array was used to keep track of all the pages of physical memory. Each of the various elements of the array were chained to various lists depending on the current use of its associated page. List structures for queues were operated on a first in - first out basis. The event list was maintained in chronological order so that the next event to occur was always at the top of the list. Entries in the event list consisted of paging and I/O interrupts and terminal interrupts. A separate location was used to represent the time of the next task interrupt. Thus the next event to occur on the system was determined by a comparison between the entry at the top of the event list and the next task interrupt.

The model was implemented in FORTRAN. It consisted of approximately 7,000 source statements organized into 31 subprograms. All major variables were allocated to a common data area. Because of storage restrictions the original model was set up as a series of overlays.

The simulator was capable of simulating as many as four CPU's, four paging drums, four paging disks, and as many as 50 other I/O devices. A maximum of 234 tasks were allowed on the system at one time and a particular task's maximum size was 128 pages. A more complete description can be found in [41, 42].

This chapter has identified the area of research, scope, and objectives. Justification for the work was presented. A brief historical review was provided along with a discussion of some of the major problems of system performance. An outline of a computer system

simulator was provided to familiarize the reader with the general concepts of computer system simulation and to serve as a basis for an understanding of the methodology. In the next two chapters the discussion turns to the problem of analyzing and classifying simulation models.

CHAPTER 2

ANALYSIS OF MODEL STRUCTURE AND CHARACTERISTICS

2.1 Fundamental Concepts

This chapter restricts the scope of the thesis further by excluding all types of models from consideration except procedural models. Procedural models refer to the class of models that are implemented by means of a computer program. It defines structural and operational characteristics and provides examples of the three different types of procedural models.

As has already been indicated, performance evaluation of computer systems is a very wide field of study. There are several ways of evaluating the performance of a computer system, simulation is just one of them. Another method which has found wide applicability recently is the technique of performance monitoring. Pinkerton [45] used a software data collection facility to record system events for later analysis. Cockrum and Crockett [10] used a hardware monitor to investigate the performance of a system. Undoubtedly the use of this technique will increase since it is a valid technique that can be used with relative ease.

Representative programs or benchmark programs can and have been used to compare the performance of two or more distinct systems or to evaluate the performance of a particular system operating under

diverse policies. Lucas [29] outlines other valuable performance evaluation techniques.

Simulation has a place too. The reasons for and advantages of simulation were outlined in Chapter 1. It is to this aspect of performance evaluation, simulation, that attention is now focused. Simulation means modelling and as such includes the full range of techniques for building models of computer systems. The following diagram is helpful in understanding the relationship between the various types of simulations.

COMPUTER MODELLING AND SIMULATION

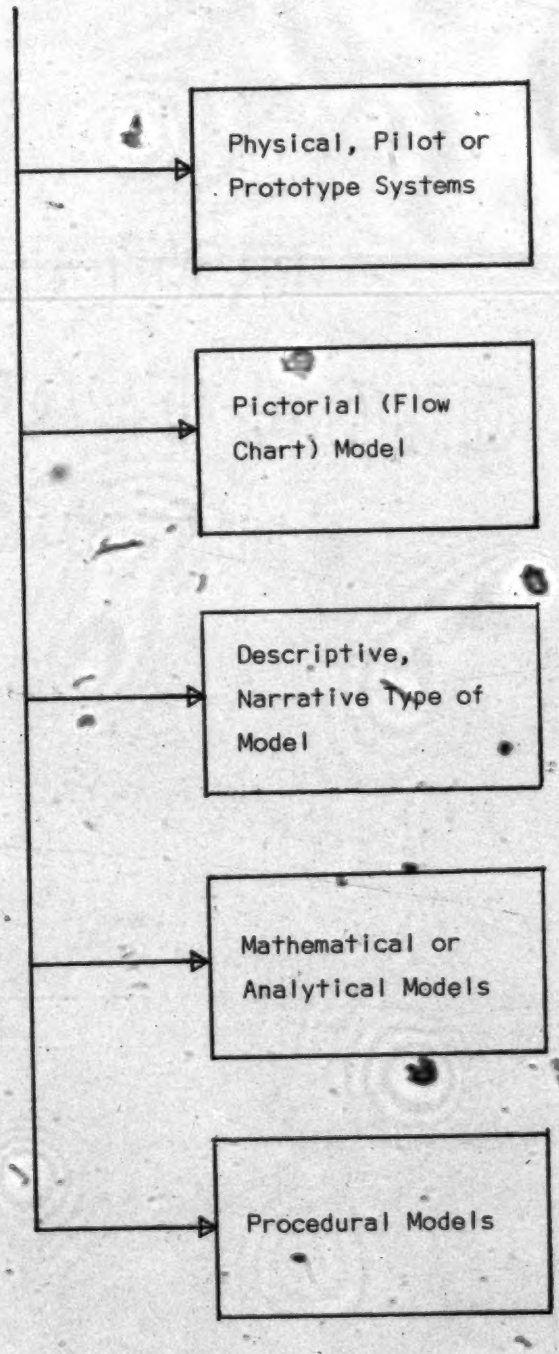

Physical, Pilot or Prototype Systems
Pictorial (Flow Chart) Model
Descriptive, Narrative Type of Model
Mathematical or Analytical Models
Procedural Models

Figure 2

Physical models of computer systems are very seldom employed because of the extreme cost. Except for development purposes where new architectural concepts are being investigated by the production of a prototype system, actual physical models of computer systems are not practical. Pictorial and descriptive models would suffice for simple problems, but the analyst would become lost when attempting to apply this technique to a complex computer system and then trying to use the description to ascertain how the system performs.

Mathematical or analytical models find practical application in the solution of types of problems such as a system of linear equations, a system of partial differential equations or complex queueing systems. It has been shown that results are quite satisfactory even for relatively complex situations. However, such models sometimes have limited applicability and lack the flexibility to permit a number of different model algorithms to be investigated with a minimum of effort.

Procedural models refer to that class of models that is represented by a computer program and that operate on an event basis. That is, the flow of the model proceeds from one event to another where each event causes some specific action in the model. There are a considerable number of such models in existence today (see Chapter 1). Each one has its own particular characteristics and mode of operation, etc., that are peculiar to its own environment. The large number of models in existence today results from the elimination of the programming bottleneck of the late sixties by the introduction of specialized simulation languages such as General Purpose Simulation System (GPSS). It is also because of the increased use of interactive simulation such as that provided by the interactive language On-line Programming System - 3 (OPS-3).

Procedural models themselves may be further subdivided into digital, analog and hybrid models. This thesis deals only with digital procedural models. The next section analyzes the structure of these procedural models of computer systems to identify common structural elements and operational characteristics.

2.2 Analysis of Model Structure

Looking at various procedural models of computer systems leads to the formation of a basic structure of these types of models. This structure is outlined below. There is an abundance of these types of models that exist today and it is useful to look at a number of these to gain an understanding of how these models operate. Representative models from each category are examined in some detail since most models conform to one of these categories. It would be too time consuming to examine all the models that have been produced.

The characteristics of models can be divided into two distinct types based on the model design considerations. The two distinct types of characteristics are structural and operational. Structural characteristics are those that are defined during the design of the model, i.e., before implementation, whereas operational characteristics really have little bearing on model design but do depend on the implementation of the model. An analogy can be drawn to the construction of a building. Structural characteristics would refer to whether the building was made of wood, concrete, steel, etc.; operational characteristics would reflect the installation of elevators as well as stairs. The distinction is necessary to define the characteristics of models.

2.3 Event Oriented Models

2.3.1 Neilsen's Simulator

The simulator has already been described in considerable detail [41,42] so it is necessary only to re-examine the model from the point of view of structure. In order to be responsive to changes in hardware configuration the model is highly parameterized. To be able to test new software algorithms, they are isolated as much as possible, or in other words, a modular approach is used. The job mix is set up semi-deterministically. Facilities are included in the model to handle multiprocessors. This means the model has to represent parallel operations in a serial manner. Paging, input/output and terminal interrupts are all placed in a chronologically sorted queue so that the first item on the queue is the next event to occur.

2.3.2 McDougall's Simulator

McDougall's model [35] is relatively simple in terms of scope, however, it is useful for outlining basic model structure. The basic operation of the model is as follows: When a job arrives at the system, it requests CPU time and memory. If none is available the job is placed in a queue. When the CPU and memory are free, execution continues until an input/output request occurs. The direct access storage device processes the input/output request unless it is busy while the CPU devotes time to another job. When a job is completed its memory is released and it leaves the system.

This model also displays modular characteristics insofar as it is subdivided into an event routine, an initialization routine, and a scheduling routine. Job mix generation is stochastic. Job

characteristics are generated from appropriate statistical distributions. The simulator handles overlapped input/output. The model is event oriented. Events are entered into a chronologically sorted queue and the first item on the list becomes the next event. The clock is advanced to that time and the event triggers certain actions.

2.4 Trace Driven Models

2.4.1 Cheng's Trace Driven Simulation

In the trace driven approach to computer simulation, modelling data is traced on a real, running system and used to drive a model. This approach reduces the work involved in specifying the workload and to a great extent the operating system too. A gross model of the operating system is still required and the real computer system must be operating.

Cheng [8] outlines a trace driven model. Two sets of input data are used. The first is the system configuration and the second is the trace data resulting from running the jobs on an existing system. Actual trace data consists of a record of all requests for transmission of data to and from an input/output device and the corresponding completion of the input/output event. CPU processing times are also recorded as well as some system functions. A job consists of a series of computing segments. Computing segments consist of CPU processing time and some associated input/output activity. A further distinction is made between overlapped and non-overlapped CPU processing. The routine of a job then depends on the duration and number of CPU processing times and input/output activities.

The data recorded represents the job profile. This job profile is reduced to cut down the volume of data and to select the data required for the simulation. The actual model is made up of several units that perform the control, operational or housekeeping functions. When an event occurs, appropriate changes take place in the units to reflect the new status of the system. The primary function of the model is to time the CPU-input/output events executed and terminated, depending on the availability of resources and operational specifications, in accordance with the scheduling and dispatching procedures.

The model records data on the starts, stops, and elapsed time of all jobs. It also collects and records data on utilization for the CPU's, main storage, input/output devices, etc. Queue statistics are also provided.

2.4.2 Noe and Nutt's Trace Driven Simulation

Another example of a trace driven simulation was presented by Noe and Nutt [44]. The simulation deals with the Control Data 6400 computer. The general approach involves the development of a simulation at a level of detail that shows the interaction between jobs and resources. Trace data is obtained from an accounting file and used to describe the job mix. Each job is treated as an individual entity. Parameterization allows choices in representing a range of operating systems and hardware components. The model is built on a high level of detail and its general structure resembles the BASYS simulator of McDougall described earlier. The model is programmed in FORTRAN. It requires approximately 70 seconds to simulate 1 hour of real time. The model provides data on turnaround time, time spent in input/output

queues, etc. Basically the model describes a series of stages such as input, processing, and output, and then identifies the conditions necessary to pass a task on to the next stage. The process continues until the job is completed.

The job characteristics are specified by 19 input parameters that describe memory used, CPU time used, cards read, lines printed, etc. Output consists of statistics on the jobs run, queues and hardward utilization. Validation of the model consists of comparing the output of the simulation to real world data. No experimentation with various job mixes and hardware configuration was reported in this paper since the purpose of the exercise was to provide and validate a trace driven simulator. The next section examines the third type of simulator, the generalized model.

2.5 Generalized Models

This third category of models deals with the so-called generalized or empirical models. This category includes systems such as SCERT (Systems and Computer Evaluation and Review Technique), CASE (Computer-Aided System Evaluation), S.A.M. (System Analysis Machine), LOMUSS (The Lockheed Multipurpose Simulation System), and ECSS (An Extendable Computer System Simulator) as general examples. These models permit ease of use if their structure fits the particular system. Models such as these can be used to determine memory size, peripheral terminal connections, operating system allocation rules, etc.

SCERT is a family of programs that can be used to simulate the performance of some set of processing requirements against a specified set of hardware. Software and hardware characteristics are kept

in a constantly updated tape library. The first phase of the system is used to describe the environment of the model. That is, unique processing requirements for each individual computer run are described in terms of the system, environment and files. During the second phase SCERT's library supplies the hardware and software performance factors of the particular configuration specified. The third phase performs numerous pre-simulation algorithms that allow the input specified in the first phase to be described in conjunction with the hardware/software system so that the same processing can be modelled on widely differing systems. Basically this phase determines input/output timing and processing times. The fourth phase actually performs the simulation to determine throughput, simultaneity and concurrency in multiprogramming and multiprocessing environments.

The fifth phase produces management reports at various levels of detail. A brief description of SCERT is provided by Herman [20]. An excellent and more complete description of SCERT has been provided by Joslin [22]. Basically the system operates on a table lookup basis to determine input/output and processing times, then simulates the flow of these timings through the specific configuration.

These generalized simulators are highly dependent on prior existence of hardware and software. It is difficult to assess the effects of multiprogramming, time sharing or terminal operations using these models. From these illustrative examples and other models studied, three basic types of models have been identified. These are event oriented models, trace driven models and generalized models.

The characteristics of these types of models have been presented and in Chapter 3 these characteristics are organized and used to classify models.

CHAPTER 3

CLASSIFICATION OF MODELS

3.1 Basic Model Structure

Having looked at the basic models, their features can be described in terms of the structural and operational characteristics defined in Chapter 2. This chapter outlines a covering set of characteristics and uses this set of characteristics to classify procedural simulators. What are the structural characteristics of procedural models?

The first is the way a model changes from state to state. Events can take place in a continuous manner; variables can take on any real value in a prescribed interval or intervals. Systems like this are called continuous systems. Discrete systems contain variables that take on only particular values. In general, models of computer systems are discrete systems. However, some models are of the continuous type. Many simulation languages exist that allow the user to program his model in either a continuous or discrete manner. A summary of discrete simulation languages has been presented by Kay [26]. For a more complete list of simulation languages see Sammet [49]. Recently languages and models have been developed which reflect a combined continuous/discrete approach. One such example is reported by Reitman [47].

The second refers to the nature in which the output is related to the input. The output of a deterministic system can be predicted completely if the input and the initial state of the system are known,

whereas in a stochastic system it is possible only to predict the range within which the output will fall. A model of a computer system is stochastic insofar as for a given work load, configuration, and system model, the resulting computer utilization cannot be predicted. Stochastic system employ approximations where exact knowledge is lacking or ill-defined. They employ statistical and/or random number generating techniques to provide the element of randomness within the model.

The third is the method used to drive the model. Several methods have been outlined already. SCERT, for example, is driven by a library of tapes containing tables that provide characteristics of equipment and software. Another popular technique is to trace the flow of a job mix as it is run on an existing system, edit and consolidate the trace data in some way to describe the characteristics of each job independently of the system they were run on, and use this data as input to a model of a system that can be modified at will.

Time driven systems refers to those systems that advance or progress through a series of states controlled by time. This type of model can be subdivided into event oriented and time slice models. Event oriented models proceed from event to event until a prescribed sequence of events is completed. In time slice models, the status of the system is updated in fixed time increments until a predetermined time has elapsed. Event oriented models are easier to produce but the approach may not be easily applicable to systems in which events may occur simultaneously.

The fourth, an important factor in the structure of computer simulation models in today's era of on-line programming, is the

development of an interactive model which allows the modeller to instantaneously adjust the model and perform another simulation. One might argue that this capability can be included or excluded from a model and is a function of the implementation. This type of model adds another dimension to simulation and as such affects the design of the model. Factors that affect the design phase of the model are considered structural characteristics. Typical interactive simulation languages are OPS-3 and Interactive Simulation Lanugage (ISL).

There are other characteristics of models but these can be classified from a programming or implementation point of view. These characteristics are called operational characteristics and they are described in the next section.

3.2 Identification of Operational Characteristics

Besides the structural characteristics which define how a model is constructed there are a number of other characteristics that can be considered useful or desirable in a computer simulation model. In this section an answer to the question, "What are the features of models that are implementation oriented?" is sought.

One of the most important operational characteristics of a simulation model is that it be modular in design. A modular design means that the simulation model is broken down into pieces or parts that are functionally different. That is, each module of the model performs a different function. This approach has the advantage of decentralizing the model so that it is in a more manageable form. Modularity allows for ease of modification. If different operating system algorithms are to be explored in a particular model, each

algorithm should form a separate module so that replacement can be made with minimum reprogramming. Care must be taken, however, to correctly specify the links between modules and, where necessary, to take into account interactions between modules.

All of the models studied employ this concept to segregate the functions of the simulator, to reduce the overall simulation package into manageable portions, and to provide for ease of implementation. Most models of computer systems are divided into at least three basic phases. These are a job definition phase, a configuration definition phase, and an operating system phase.

Parameterization is another important characteristic. Computer simulation models generally digest vast quantities of data and if rigid input becomes the rule then the scope of the model is severely restricted. Every effort should be made to specify variables that can take on a wide range of input values as parameters. In this way the model becomes considerably more flexible allowing for the simulation of a wide variety of job mixes, for example, or hardware configurations.

Models should be well documented, a simple requirement but frequently overlooked or poorly organized. The documentation for a model should be such that someone with only a general understanding of computer systems can prepare, make a run, and evaluate the results from a run. The best way to accomplish this objective is to keep the input required as simple as possible and to produce clear and concise system performance data. This would depend to some extent on the level of detail of a model. Detailed models require more sophisticated input and as a result are more complex.

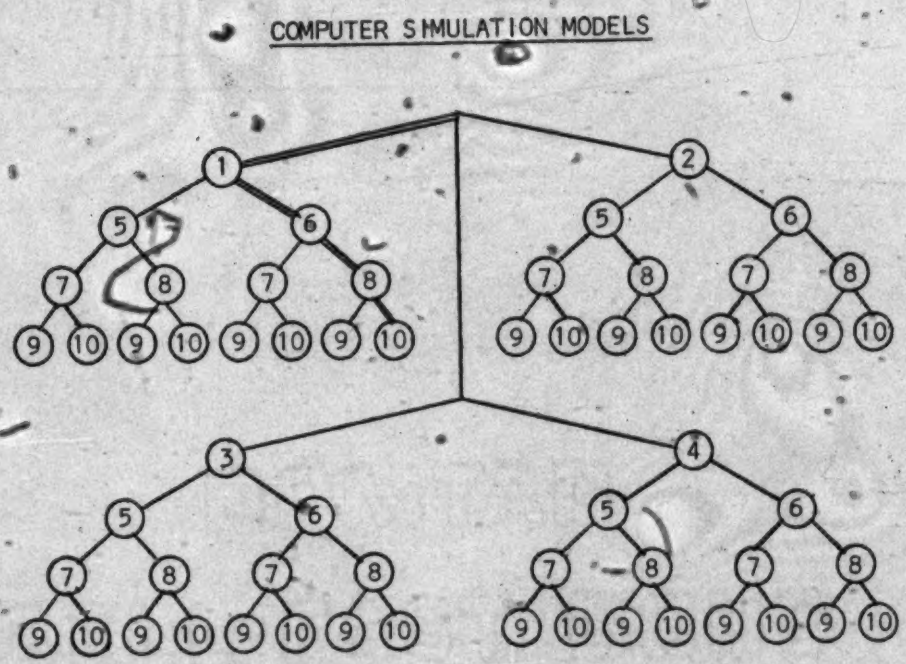
The level of detail of a model is an important item. Ideally, a model should be designed so that different parts of the system can be simulated at different levels of detail. The major problem with the level of detail is to decide just how much detail is required to get accurate results. The level of detail may range from two extremes. There is the micro level of detail. Simulators can be constructed that model system operations in minute detail. On the other hand there is the macro level of detail where simplifying assumptions are used to reduce complexity. The ideal situation is to keep the level of detail as high as possible, that is, at the macro level.

System performance statistics should be available so that an analyst can determine how the simulated system operated. System performance statistics should be concise. That is, few people like to analyze enormous quantities of data; and summary reports, indicating the major performance criteria, should be the analyst's objective when the output reports are designed.

3.3 Classification of Models According to Characteristics

In this section a classification scheme is outlined for procedural models of computer systems based on the four fundamental structural characteristics described in Section 3.1. Models can be classified, first of all, by model type. They may be event oriented, trace driven, generalized, or time slice models. Time can advance in a model continuously or in discrete intervals. Job generation can be deterministic or stochastic. Finally, a model may be designed to run in a batch mode or in an on-line mode.

The following tree structure can be used to show the relationship between these four items.



1. Event-Oriented
2. Time-Slice
3. Trace-Driven
4. Generalized
5. Continuous
6. Discrete
7. Deterministic
8. Stochastic
9. On-line
10. Batch

Figure 3

Using this scheme all 32 different combinations of characteristics can be described. A sample is shown by the double line connecting nodes. The type of model represented by the double line is an event oriented, discrete, stochastic, batch system. Not all branches are utilized. For example, trace driven models are generally discrete, deterministic, batch models. All of the procedural models studied can be classified according to this scheme.

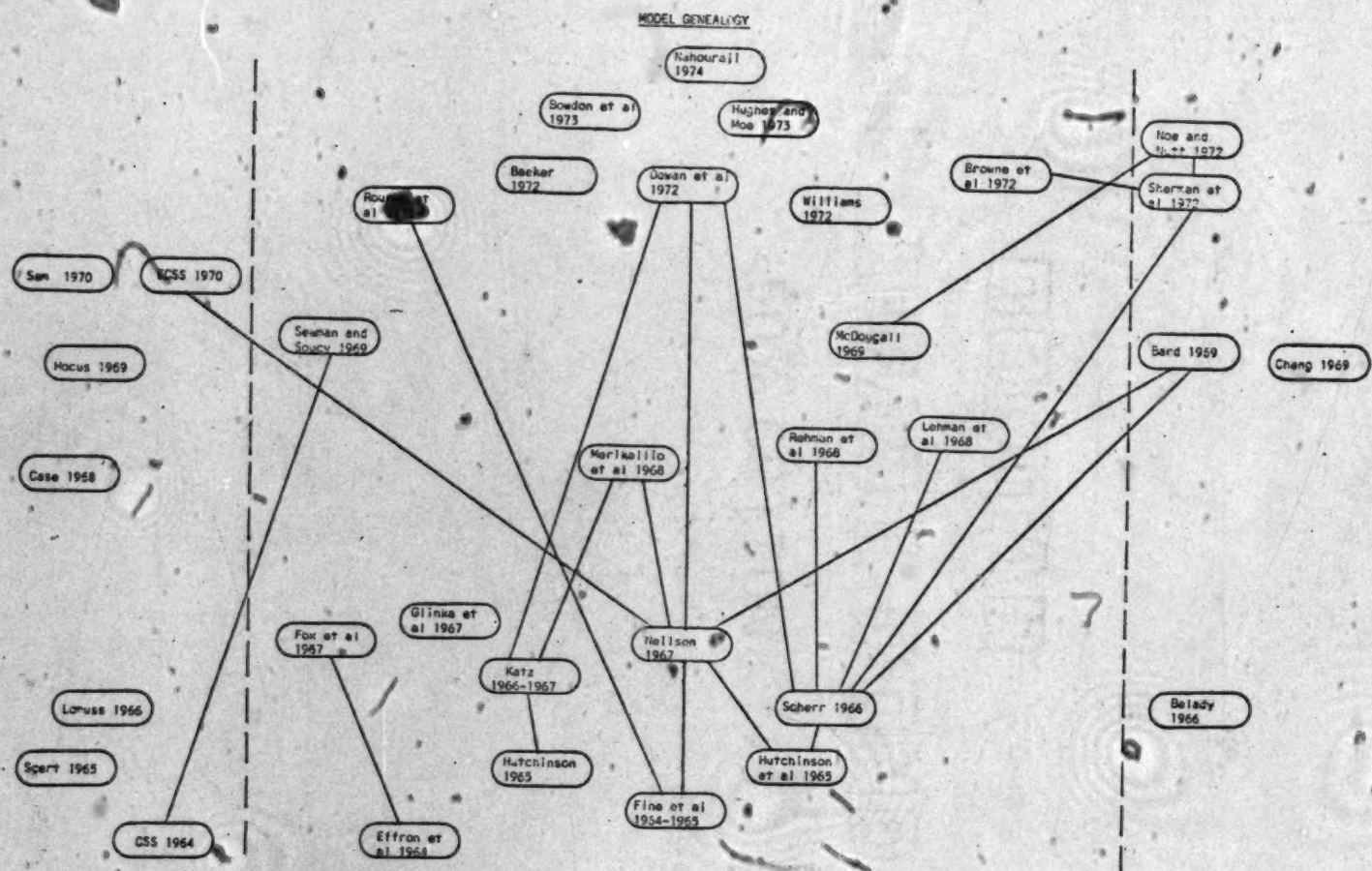

MODEL GENEALOGY
Nahouraji 1974
Bowdon et al 1973
Hughes and Moe 1973
Noe and [illegible] 1972
Browne et al 1972
Sherman et al 1972
Becker 1972
Dowan et al 1972
Williams 1972
[illegible] et al [illegible]
Sam 1970
ECSS 1970
Seaman and Soucy 1969
McDougall 1969
Bard 1959
Chang 1969
Hocus 1969
Case 1968
Markwallio et al 1968
Rehman et al 1968
Lehman et al 1968
Glinka et al 1967
Fox et al 1967
Nelson 1967
Katz 1966-1967
Scherr 1966
Belady 1966
Larsen 1966
Spert 1965
Hutchinson 1965
Hutchinson et al 1965
Fine et al 1964-1965
CSS 1964
Effron et al 1968
GENERALIZED MODELS
EVENT-ORIENTED MODELS
TRACE DRIVEN MODELS

Figure 4

When combined with the operational characteristics, this scheme forms a complete classification scheme for procedural models.

The diagram on the previous page shows the relationship between procedural models of computer systems. This structure contains a map of the majority of procedural models that were found in the literature surveyed. It is organized by year to show the development of models and by model type to show the classification of models.

In the two previous chapters the fundamental concepts of computer model building have been presented. In particular, the scope of the thesis was restricted to digital procedural models of computer systems. Several models were reviewed in detail to determine their structure. From the data gathered definite structural characteristics were identified and computer simulation models were classified according to their structure. Other operational characteristics were isolated and described. It now remains to use these structural and operational characteristics in the development of a methodology for building these types of models and show how the methodology can be applied.

CHAPTER 4

DESCRIPTION OF THE MODEL METHODOLOGY

4.1 Introduction

In this chapter the model methodology is presented. A notation for describing prototype jobs is outlined in the description of a job mix generator. A method is specified for defining hardware and an operating system. Criteria, which can be used for the measurement of system performance, are discussed. The methodology allows an analyst to simulate various subsystems at different levels of detail.

From the analysis of model structure four basic and necessary parts are defined. These are a job generator, a hardware configuration, an operating system, and system performance statistics. The relationship between these modules can be seen in Figure 5. This chapter is devoted to the definition of each of these sections with particular attention being paid to the level of detail.

MODEL METHODOLOGY

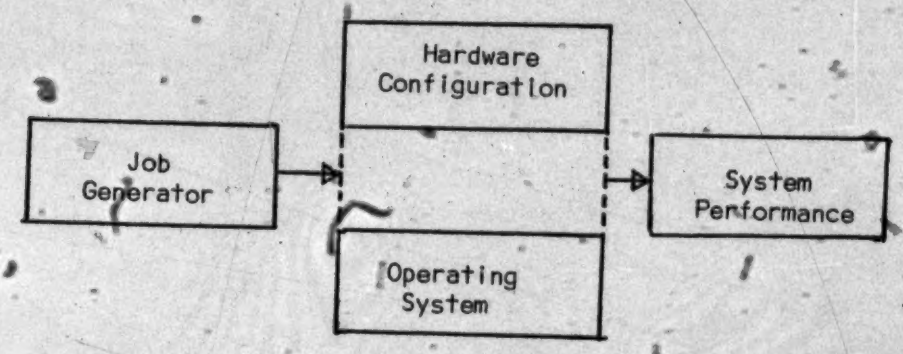


Figure 5

The job generator prepares a job mix that is to be run against a particular hardware configuration as defined by the hardware configuration module. The job mix is processed on the specified hardware according to the operating system described by the operating system module. Results are gathered and reduced by the system performance module to provide a meaningful analysis of system performance.

4.2 Job Generation

A major problem that must be dealt with in any computer simulation model is the representation of the work load or job mix. Many articles have been written on this subject but no satisfactory solution has been put forth. However, a number of approaches have proven worthwhile and since these approaches are interesting and do reflect different lines of attack on the problem, some of them are discussed below.

Lucas [29] presented a definitive article on performance evaluation and monitoring. In that article he recommends that primarily for the evaluation and selection of new computer systems, the synthetic program is the most attractive technique from the point of view of cost and capability. Each synthetic program would consist of a series of modules that combine to form a program. A number of these programs would be produced in a representative mix.

Kimbleton [27] stresses the use of hardware and software monitors as a means for measuring the effects of the job stream upon a given computer system. This technique requires an existing system and the subsequent analysis of trace data to determine job characteristics is difficult. Data derived from monitors is most useful in improving performance by allowing significant configuration improvements and by

optimizing frequently used software.

Bard [3] describes a system of job generation using synthetic benchmarks. A set of representative jobs is set up whose characteristics are readily altered by simple adjustments to parameters. The technique is most useful for analyzing new systems and can be used to evaluate changes to existing systems provided the benchmarks are run before and after the change.

One of the most aggravating problems is at what level of detail should the jobs be represented. A wide range of detail can be used to describe any particular job. The full detail in terms of the number and types of machine language instructions can be specified. This is the micro level of detail. On the other hand job specifications can be described in a highly parametric and stochastic fashion. This is the macro level of detail. Trade-offs have to be made in any simulation between the accuracy obtained at the micro level of detail and the ease and speed with which jobs can be described and modified at the macro level of detail. Since the simulation effort is directed at establishing trends, gaining insights, etc., rather than exact prediction, the representation of the job mix in the methodology tends toward the macro level of detail.

Rather than trying to develop job mixes that will provide guiding principles for the synthesis of configurations or strategies that manage resources, the job characteristics should provide a reasonably accurate representation of program rates and volumes. The results from the simulation model itself should determine the former.

Three questions must be answered about jobs. What parameters should be used to characterize a job? What values should be assigned

to these parameters in order to duplicate effectively the job mix that is to be modelled? What is the distribution of arrival times of the various jobs? The choice of parameters is to some extent dependent upon the objectives of the simulation, but a basic set of parameters can be defined. These are defined below:

1. Job Identification
2. Input Function
3. Compute
4. Output Function
5. Size

Job identification simply refers to some method of labelling jobs. Input function refers to the input that takes place in a program or job. Similarly the output function refers to job output. Compute defines what computation has to be carried out by the program. Size describes the amount of memory occupied by the program. An example is shown below.

#1 50 INPUT 10,000 COMPUTE 100 OUTPUT 4K

This representation describes a very simple job that inputs 50 records, performs calculations supposedly on the input that account for 10,000 computational units and outputs 100 records. The program takes up 4,000 memory locations. Note that no specific I/O devices are assumed and no particular time has been assumed for a compute unit or I/O unit. In a simple environment it does describe the job well, and in conjunction with the appropriate hardware and operating system specifications, it can be used in a simulation. Unfortunately, today's computer system is not as simple and other parameters and a more complicated job structure need to be added to adequately describe jobs that run in such complex

environments. However, the fundamental structure of this unit will remain the same. The remaining discussion deals with representing more complex jobs using this structure.

File input/output (I/O) is represented by specifying the device followed by the word "INPUT" or "OUTPUT". Words such as "READER" and "PRINTER" can be used where there is no ambiguity with respect to input or output.

Another characteristic is required to describe on-line jobs. That characteristic is user think time, where user think time is defined as the time spent while the user types in information, receives information, thinks, etc. Boies [5] has compiled excellent statistics on this subject.

To allow the simulation of jobs run in a virtual memory environment (regardless of how the virtual memory is implemented) a characteristic which has been called a page fault is required. One more characteristic is required to allow for the manipulation of files, in particular, long term storage files. The file command allows the simulation of events such as file create, file destroy, etc. This facility is intended to describe the situations where a user program manipulates files. It is not intended to represent system file handling. For example, the creation and destruction of spooling files. These two characteristics can be grouped under one event or type of interrupt which is called a memory interrupt. This new set of characteristics is shown below:

1. Job Identification
2. Compute Function
3. File I/O (Disk Input, Tape Output, Terminal Input, etc.)

4. Think Time
5. Memory Interrupt (Page Input, Page Output, File Create, File Destroy, etc.)
6. Size

This set of characteristics allows an analyst to specify almost any type of job. Indeed, if for some reason other characteristics are required to describe a job they can be easily added to this fundamental set. One such characteristic might be a protection fault to simulate memory references to unauthorized areas by user programs.

With this set of characteristics how is a complete program represented? Since programs are complex and composed of many separate steps, the concept of phases within programs or jobs is introduced. A program may consist of one or more phases. The example used before to describe a simple input/compute/output job was a one phase program. The next example represents a batch FORTRAN compile.

#1 100 (5 READER; 15,000 COMPUTE; 10 PRINTER; PUNCH)

This describes a complete program that consists of four operations. Operation 1 represents reading the deck, Operation 2 represents the computation performed during compilation, and Operations 3 and 4 represent the listing of the source deck and the punching of an object program. The numbers preceding each operation represent the number of repetitions of that operation. The number preceding the phase represents the number of repetitions of the phase. For example, this program consists of 1 phase repeated 100 times.

The construction of the job is operating system independent. This means that functions, such as spooling, are part of the operating system and jobs need not be described according to the nature of the

operating system. Files for spooling would be automatically assigned by the operating system model if spooling was in integral part of the operating system. A terminal/file edit type of job is shown next.

#2 5 (TERMINAL INPUT; 10 COMPUTE; TERMINAL OUTPUT)
(TERMINAL INPUT; 20 COMPUTE; TERMINAL OUTPUT)
25 (TERMINAL INPUT; 10 COMPUTE; DISK OUTPUT; 10 THINK TIME)
(TERMINAL INPUT; 100 COMPUTE; 10 TERMINAL OUTPUT)
25K

Phase 1 is the signon, Phase 2 is activating the file, Phase 3 is updating the file, and Phase 4 is the signoff. The program size is 25,000 memory locations.

#3 1000 (10,000 COMPUTE; TAPE OUTPUT; PAGE INPUT)
100 (OUTPUT; 500 COMPUTE) 10K

Here we have a job with a heavy computational load and little I/O. A page fault has been included to show how that characteristic is used. A formal notation has been defined for describing jobs. An operation is defined as:

$R_i O_i$ where R_i is the number of repetitions of operation i and O_i is the i^{th} operation within a phase.

A phase (P_i) consists of a number of operations:

$$P_i = (R_1O_1; R_2O_2; \ldots ; R_nO_n) \qquad n \geq 1$$

A phase may be repeated several times. This is indicated by a phase repetition number, PR_i.

$$P_i = PR_i(R_1O_1; R_2O_2; \ldots ; R_nO_n)$$

A job consists of a number of phases.

$$J_i = P_1P_2 \ldots P_n \qquad n \geq 1$$

Using this notation an analyst can describe the job mix to be modelled. Two points are worth stressing again. They are:

1. The job characteristics are flexible. That is, there is no need to be restricted to the set used in the examples provided.
2. The description of a complete program can be done in as much detail as desired. Each separate part of a program can be described as a phase, or a program can be represented at the macro level of detail simply by one phase.

The structure described above details what happens in a program but no mention has been made yet of when event happen. Question two was, "What values should be assigned to these parameters?" There are three basic ways in which values can be attached to each operation. The values may be fixed, selected from appropriate distributions, calculated from hardware/software characteristics, or some combination of all three may be used. For example, the basic "COMPUTE" unit could be fixed at 10 microseconds. "THINK TIME" could be derived from a statistical distribution, and file I/O might be a combination of timings that take into account device transfer rates, access times, and an estimate of software overhead time.

Fixed units lead to deterministic (or semi-deterministic) timings for different jobs. The use of stochastic parameters would provide a much greater variability in job behaviour. The analyst though is free to choose whatever approach best describes the system he is modelling.

At this point isolated jobs have been described. To properly develop a job mix for a simulation, it is necessary to construct a set

of prototype jobs (or tasks) that would be representative of the work load to be modelled. This is accomplished by selecting a set of prototype jobs and describing them in the notation just outlined. Note that prototype jobs need only be described once. Variable job mixes are obtained by selecting different prototypes and by changing job arrival times. Once a set of prototype jobs has been set up it merely remains to define the arrival time by a suitable statistical distribution to answer Question Three, "How is the arrival time of a job specified?"

Empirical distributions should be used to describe inter-arrival times. Poisson and exponential distributions have been used in some models, however, most computer systems are subject to busy and slow periods both in the short term and long term. Empirical densities are required to match such patterns.

Processing programs can be divided into three main types. They are language processors, utilities, and user programs. The analyst, by analyzing how each type works, can describe all three different types using this notation. The prototype jobs can be controlled and manipulated to "tune up" the job mix to match the actual work load on an existing system. A job mix for a hypothetical machine can be set up based on expected work loads. In short the approach is extremely flexible and relatively easy to use.

4.3 Hardware Configuration

The second part of describing a computer simulation model involves the representation of the hardware. Consideration has to be given again to the requirement that the hardware configuration be easy to modify. With this requirement in mind it is necessary to introduce

the concept of a device description. A device description consists of a set of parameters that completely describe the device. In much the same way that a set of parameters was used to describe job characteristics so hardware characteristics can be described by a set of appropriate parameters. For illustrative purposes the example shown below is based on the IBM 360 architecture. Besides representing the actual physical aspects of the device the relationship between devices must be specified. For example, on the IBM 360, the hardware descriptions must specify which devices are connected to which channels. A set of hardware characteristics for a magnetic disk might be:

1.	Device name	Disk
2.	Quantity	2
3.	Capacity	100M characters
4.	Rates, transfer	312K characters/second
	average seek time	65 milliseconds
	rotational delay	25 milliseconds
5.	Channel connection	3

The detail specified is again very flexible and can be altered to suit the required level of detail of the model. A complete system is outlined in Table 1. This specification is called a hardware description.

TABLE 1

HARDWARE CHARACTERISTICS

Device Name	Quantity	Capacity	Channel	Hardware Rates
CPU	1	-	-	1M instructions/second
CORE	1	256K		500 nsec. cycle time
				Page swap time = 8 ms.
SEL. CHANNEL	2	8 CNTRL. UNITS	-	1.3M characters/second
MUL. CHANNEL	1	4 CNTRL. UNITS	-	110K characters/second
CNTRL. UNITS	7	16 devices		same as channel
DRUM	1	4M characters	1	1.2M characters/second
				17 ms. access time
DISK	2	100M characters	2	312K characters/second
				90 ms. access time
MAG. TAPE	2	-	3	60K characters/second
READER	2	-	3	600 cards/minute
PRINTER	2	-	3	1200 lines/minute
PUNCH	1	-	3	300 cards/minute
TERMINALS	22	-	3	22 characters/second

The analyst must be extremely careful to enter realistic figures. That is, the simulation must represent the system as it actually functions, not as it is supposed to function. Most simulation models pay very little attention to hardware characteristics. With this method an unlimited number of devices can be described and each device can be characterized by several parameters. In other words, some of the emphasis that other methods of simulation place on the operating system, by including such device characteristics in the operating

system, is removed in this methodology and elevated to a separate function of its own.

The hardware rates described in the device description can then be used by the model of the operating system to determine times for computation, file I/O, page faults, etc.

4.4 Operating System

Perhaps the most complicated and most analyzed part of a computer system is the operating system. Much research has been done on memory management, resource allocation, concurrent processing, and other facets of operating systems. It is useful to trace the development of operating systems briefly to obtain some idea of their basic structure. In general, computing systems have been used in a basic programming environment with an operating system and via remote computing facilities. Basic programming involved running each program step separately, mostly as a result of inadequate storage.

To eliminate the manual intervention and to automate the process of running a job, a computer program (or set of programs) was written. This program or set of programs came to be known as the operating system. The steps present in the previous environment also existed in the operating systems but they were monitored now by the system itself. This characteristic allowed jobs to be submitted in batches since the transition from job to job was automated.

The use of the data channel or separate I/O processor changed things dramatically. Most significantly it opened up the area of I/O - compute overlap and research was begun to explore ways to put this feature to efficient use. This research lead to the development of

operating systems that were the forerunners of today's systems. Since these systems tended to be I/O bound, the extra CPU time available could be used to run other jobs in the system. Consequently concepts such as multiprogramming, where more than one program resides in main memory, and time sharing, where the CPU is switched rapidly from one job to another, were introduced. These capabilities led rapidly to the use of terminals and remote computing.

To provide these services the operating system is usually divided into several separate and distinct areas with different functions. From a review of operating systems, four major units can be identified and for the purposes of this methodology, the operating system has been divided into these four major categories. They are:

1. Job Initiator and Control
2. Memory Management Scheme
3. Scheduling Routine
4. Input/Output Processor

In the next few pages the basic function of each of these units will be described. It is necessary to remember, however, that the actual definition of what each of these routines does depends on the type of operating system being modelled. That is, when specifying each of these four parts, they should reflect the characteristics and structure of the system being modelled. The level of detail is limited only by the constraints of the analyst. The description of the I/O processor, for example, can be at either the micro or macro level of detail.

4.4.1 Job Initiator and Control

The basic functions of the job initiator and control routine are to determine when each job is to start from statistics provided about job arrival times and to initiate execution of that job.

Initiation of execution generally involves loading a program into main memory or queueing the program if memory is not available. Once a program is loaded control can be passed to the first operation of the job and the job begins its execution. In addition to initiating a job this routine also has the responsibility of seeing that control passes smoothly from one job to the next. When a new job arrives at the system this routine accepts the job and enters it into the main stream of processing. At the completion of a job, this routine performs the necessary steps to complete the job and pass control to another job.

4.4.2 Memory Management Scheme

The memory management scheme can be divided into management of short term storage and long term storage. Short term storage refers to the storage involved in processing jobs, that is, main memory, paging drums, paging disks, etc. Long term storage refers to that storage area allocated to data files, program modules, etc., that are retained past the completion of a job. The analyst has the responsibility to describe the memory management scheme according to how the operating system actually performs. The memory management scheme usually takes the form of an algorithm that seeks to make the most efficient use of the storage of the computer. Typical functions performed by the scheme are:

1. Loading programs
2. Processing page faults
3. Unloading programs
4. Assignment and control of long term storage
5. Spooling file allocation
6. Maintenance of catalogs (for example, for data files)

In a paged environment the memory management scheme must also reflect the type of implementation.

4.4.3 Scheduling Routine

The scheduling model determines which job can next be submitted to the central processing unit. Sufficient memory must be already available and the job should have been loaded before CPU cycles can be allotted to it. Therefore the memory management model must be invoked before the scheduling model. Once memory has been allocated, the scheduling model allocates the CPU to the job until some event takes place that causes the CPU to halt. These events are called interrupts. There are several types:

1. Memory
2. Input/Output
3. End of Compute
4. Protection
5. Error
6. Interrupts by Higher Priority Programs

When one of these interrupts occurs the job is "removed" from the CPU and control is passed to an appropriate routine to handle the particular interrupt that has occurred. The CPU is generally

switched to another job rather than being left idle. The manner in which the CPU is switched to another job is determined by the scheduling algorithm.

4.4.4 Input/Output Processor

The input/output processor basically communicates information to and from the outside world and between devices. The input/output processor handles all of the information that passes between the user and the computer. One subroutine should exist for each type of I/O operation. In a batch mode, where I/O is spooled, there should be a routine that processes spooled input and output. Input/output routines should exist for disk, drum, magnetic tape devices. Unit record equipment, such as a card reader or line printer, should have separate I/O routines. In general, one can say that each routine should simulate how long the device takes for I/O. The I/O processor handles all file I/O and spooling I/O. It should not handle paging I/O. The I/O processor is also responsible for determining if devices such as disk and drums have available space or are busy. If no space is available or the device is busy, the I/O request can be queued to be processed later. Maintenance of the specific device queues is also the responsibility of this routine.

Most operating systems are interrupt-driven. That is, the system continues processing until some event takes place that requires the attention of the CPU. The CPU processes the event and then returns to the original (or another) job to continue processing it. When an interrupt occurs control is passed to a routine to process the interrupt. These interrupts are termed hardware interrupts in that

they are initiated from a hardware device. They may be from an I/O device, the CPU, an electronic device such as a timer or clock, or from the system console. Another set of interrupts can be defined as software interrupts. These are typically from machine instructions. They include interrupts such as protection faults and error faults, etc. The operating system's function is basically to respond to these interrupts by identifying them as to type and by initiating appropriate routines to process the interrupts. In so doing, the operating system maintains control over the whole system. Generally, these routines, or a fundamental set of these routines, will reside permanently in main memory and are not directly accessible to the user.

It is on this basis that the parts of the methodology are connected. Chapter 5 discusses the implementation of the methodology, but first, it is necessary to complete the discussion of computer systems by examining measures of system performance.

4.5 System Performance

System performance is a large enough field for a separate study of its own, however, for the purposes of the methodology system, performance refers to measures or statistics of the way the simulated system performs. The performance of the system can be divided into three rather broad areas. These are system resource utilization, job performance, and queue statistics.

The system resources can be categorized in many ways. A useful way to examine system resources is by separating hardware and software resources and then defining primary and secondary resources within each category. The following diagram is helpful in establishing the relationship between these different categories.

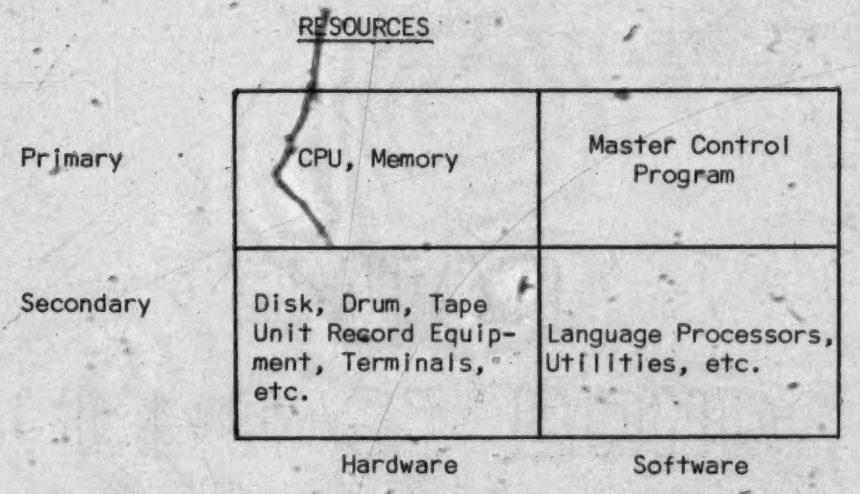


Figure 6

Utilization can be recorded for each device or program. The number of reads, writes, the number of pages read, written, etc., can all be recorded for each device. The analyst is free to define whatever performance characteristics he desires. More complete and detailed statistics are required for the primary hardware resources. For the CPU, data such as execution time, overhead, idle time and total available CPU time can be recorded to give some idea of how the CPU is performing. Paging data is very important and records should be kept of the total number of pages used, the number of pages read, the number of pages written, etc. On the software side the control program has been included as a primary resource since it provides a control function. However, time spent in this mode of operation is time that is not available for the users. Therefore the utilization of this resource must be measured with respect to how well it performs its function. That is, the more CPU cycles the operating system can deliver to the user and still provide the required services, the more valuable that operating

system is. Usage of language processors and utility programs can be recorded by counting the number of times each process is used. More sophisticated data can be collected. For example, the CPU time used by each process could be recorded.

Some measures refer to the performance of the system as a whole. It is useful to obtain some information about how the complete system is working. In an on-line environment response time could be used. Throughput, or the number of jobs processed per unit of time is another useful measure.

The second area of concern is the monitoring of job performance. Just as the system has useful measures, so do individual jobs. Job performance characteristics include data pertaining to:

1. Memory used
2. CPU time used
3. Time in queues
4. Turnaround time
5. Quantity of I/O
6. Paging statistics if applicable
7. Cost

The third area in which records and data should be collected is in the area of queues. Most hardware devices have queues associated with them so that if a device is busy when a request for that device is made, the request can be placed in a queue for subsequent processing. Statistics on these queues can lead to the identification of bottle necks. Data should be recorded on queues for all hardware devices including terminals, unit record equipment, disks, tapes, drums, central processing unit and main memory. Of particular interest in a paged

environment would be the collection of data on the page queues.

In summary, system performance can be measured by recording data on the utilization of resources, performance characteristics of individual jobs and device queues. Measures of these items must be incorporated in a model from the design phase and not added as an afterthought. They must form an integral part of the total model methodology.

The list below can be used as a guide for the construction of a model. It covers the basic steps that are necessary to build a procedural simulator within the confines of the methodology.

1. Model Conceptualization - Formulate concepts for a job mix, hardware configuration, operating system, performance statistics, validation procedures and experiments to be conducted.
2. Apply Methodology
   - 2.1 Operating System - Sketch the flow of jobs through the system. Define the interrupts and the processing required for each interrupt.
   - 2.2 Job Mix - Answer the three basic questions regarding characterization of jobs.
   - 2.3 Hardware - Decide what characteristics are to be used for each device. Set up a device description for each device.
   - 2.4 System Performance - Define the measures of system performance that are required.
   - 2.5 Construction - Build computer programs to represent the model.

2.6 Validation - Verify the model according to the procedure outlined in step 1.

3. Experimentation - Set up and conduct experiments. Evaluate results.

4. Cycle through steps 2 and 3 until satisfactory results have been obtained.

CHAPTER 5

IMPLEMENTATION AND VALIDATION

5.1 Implementation

So far the discussion has centered around the four major constituents of a computer simulation model. However, to complete the methodology it is necessary to outline how the four modules are related. This chapter describes the implementation of the methodology and goes on to define a formal validation procedure which can be used for procedural models of computer systems.

The implementation of the model methodology is based on an event oriented procedural simulator. The flow diagram on the following pages illustrates the way the methodology is to be implemented.

Basically events are generated by the job generator from the job descriptions. These events are entered into a chronologically sorted event list. This means the event at the top of the list is always the next event to occur. Events consist of interrupts. A typical set of interrupts is listed below:

1. End of program interrupt
2. End of time slice
3. Paging fault
4. Spooling I/O
5. File I/O
6. Protection
7. Error
8. Interrupt by a higher priority program

IMPLEMENTATION

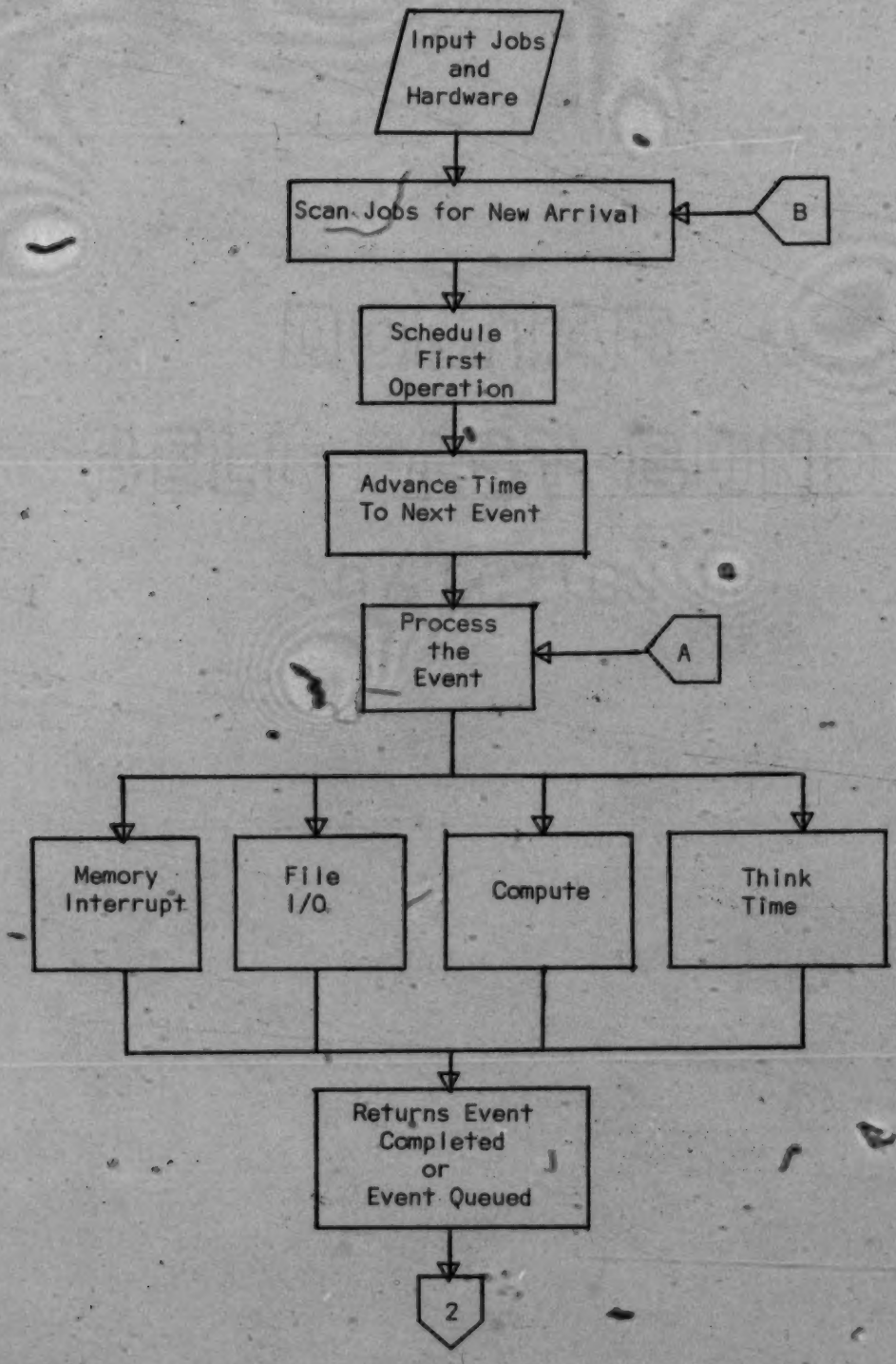


Figure 7

IMPLEMENTATION (continued)

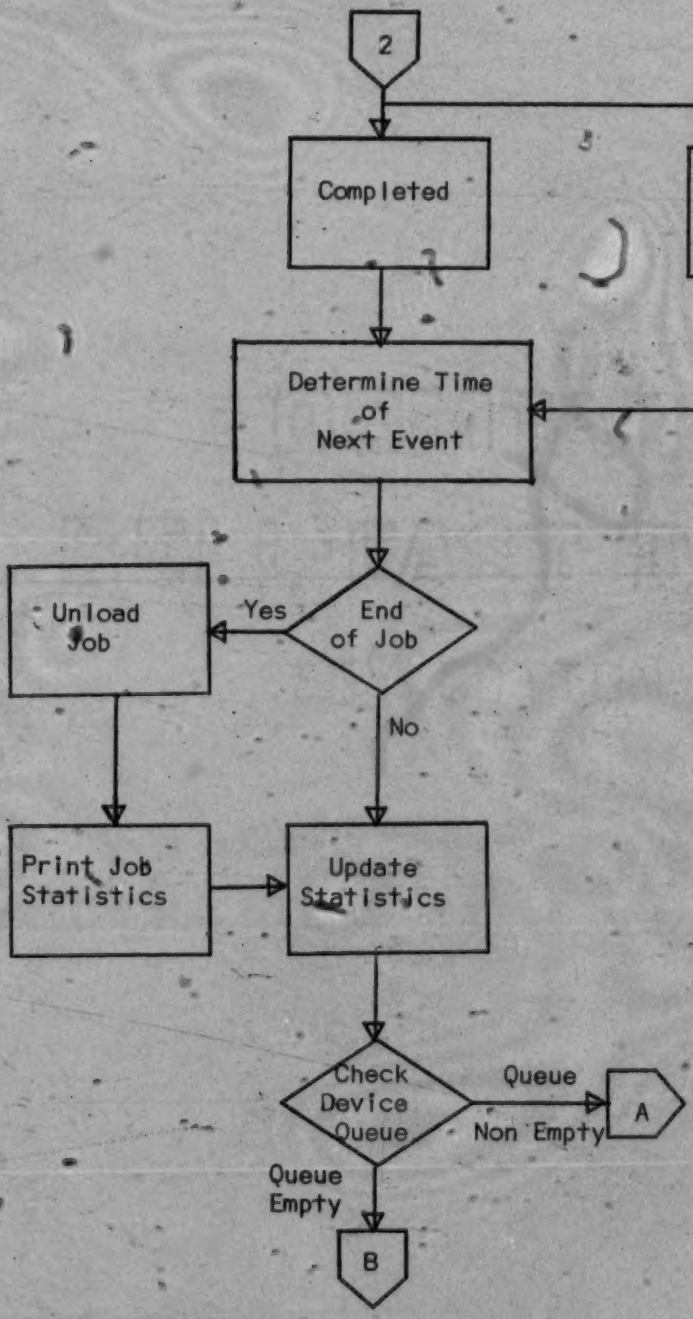


Figure 8

Each interrupt has associated with it a routine that processes that type of interrupt. One other event is included in the event list, that is the "COMPUTE" event. This event represents processing by the CPU.

The basic cycle for processing events is as follows:

1. Advance time to next event (i.e. event at top of list)
2. Process this event
3. Sequence the next event for this job

Processing continues until all the interrupts for a particular job have been completed, then that job is removed from the system. Jobs are initiated by the job initiation routine. This routine basically loads the job and sequences the first event.

A job can take on any one of three states: active, blocked, and ready or waiting. An active job is being processed by the computer. A blocked job is awaiting CPU time but cannot proceed because an interrupt is being processed. A ready job is awaiting CPU time with no interrupt pending, that is, the CPU is already busy. By tracing the status of each job, job performance statistics can be gathered. By recording the time each device is in use, device utilization can be calculated.

It is possible to have two or more events scheduled to occur at the same time. In such a situation rules have to be specified to process these simultaneous events. Where the simultaneous events refer to different devices, each event can be processed since the operation of devices is independent. However, if the simultaneous events refer to the same device, the analyst has to model the way in which the system under study handles this case.

Rather than spend a great deal of time and space on the actual implementation, which is a programming problem, it is more important to establish the validity of the methodology. The next section outlines validation procedures and proposes a formal validation technique.

5.2 Establishing a Need for Validation

No discussion of simulation is complete without reference to validation. Usually in the development of a computer simulation model, discussions take place between the analyst and the user. These early discussions lead to an initial formulation of a model concept. Differences and approximations to the real world situation are outlined. With models of computer systems; configurations, software packages, and job mixes need to be specified. When the analyst and the user are satisfied that the concepts they have developed represent accurately the system to be modelled, they can begin to validate the model and show that it does represent the real life situation. Inaccurate specifications of hardware or software in a model of a computer system could lead to inaccurate results. In short, a model must be credible.

An analyst's belief in his simulator is not enough to say it is close to reality. Only a formal validation effort can eliminate the doubt that the simulator does not represent the real life situation. Without a valid model it is useless to engage in any other activity. Without a valid model no reliable results can be generated.

5.3 Types of Validation Procedures

There are several means to arrive at a valid model. The standard validation procedure is to compare the results obtained from the model to historical or actual real life data. This is validation by comparison to actual system performance characteristics.

When no such comparison is available, validation becomes more difficult. Other methods of validation must be employed. Some of these are outlined below. One such method for the validation of models of computer systems employs the technique of substituting fixed values for job characteristics to permit comparison of simulation results with previously calculated values. That is, an attempt is made to simplify the input to the model so that values can be calculated manually prior to a simulation run and then these values are compared to the output of the simulation model. Another technique is manual verification of model behaviour through the examination of a step by step trace produced by the simulation model.

A third technique involves verifying that the simulation model follows the correct sequence of operations and correct responses are received to alterations of input. This technique and the one mentioned just prior to it are almost the same. The difference is that the previous technique involves validation inside the model whereas this technique places more emphasis on correct input/output relations.

A fourth technique is to run the model with data whose characteristics are completely known and as such will produce known results. The results from the model are then compared to the known results to establish validity.

Martin [33] comes closest to establishing a formal validation procedure. He asks, "What is a valid model?" Basically he replies by stating that a valid model mathematically and logically approximates the system under study. To achieve these goals he divides validation into two parts--validation of the conceptual model, then validation of the implementation.

To develop the model the analyst should have used correct transformations to get from the real world situation to a model. To validate the conceptual model arrived at, Martin suggests the use of a reverse reasoning process. Essentially, by reverse reasoning, examine the model, trace back to the approximations, and finally trace back to the real world. Martin suggests a step by step procedure for this reverse reasoning process:

1. Examine and review the model concept and rationale.
2. Relate to the deterministic, randomization and expected value features in the model.
3. Examine the above approximation in relation to real world.
4. Review measures of effectiveness, parameters and variables.
5. Examine assumptions and hypotheses.
6. Relate 4 and 5 to real world. Examine the system, the system environment, man and the interactions among these three.
7. Check the validity of information and data and their sources applied in the problem.

8. Review the entire procedure in relation to the problem definition.

9. Review the problem statement.

Although this procedure is not foolproof, it does afford the opportunity to re-examine the problem from a different point of view.

Just as addition is used to check on subtraction, so following through the model from end to beginning can be used to substantiate the conceptual model. He also suggests that it is sometimes a good idea to have an outside authority review the model design. With these checks the analyst is reasonably sure of arriving at a bona fide conceptual model.

Once a genuine conceptual model is available the certification of the implementation can take place. Basically, the implementation is validated by corroborating in turn, the logical flow chart, the program flow chart, and then the computer program. Each succeeding step is checked by comparison to the previous stage. For example, the logical flow chart is validated by comparison to the conceptual model, etc. The final step consists of using sample problems with results calculated independently of the model. The sample problems are run on the model and the results are compared to the independently calculated results. If possible, use of real world inputs should be made and the results can then be compared to the real-world outputs.

5.4 Formal Validation Procedure

Validation of this methodology can be tackled from the point of view of verifying each individual major module and correctly specifying inter-module interactions. Validation of the definition of the

job mix, hardware characteristics and operating system does, to some extent, depend on the use to which the simulation will be put. If the simulation is intended for selection, evaluation, or projected design, probably no actual work load exists. Work load will have to be developed from another comparable system or else from one's own estimate based on applications to be run on the computer. Hardware characteristics will exist when evaluating existing systems and can be derived from specification manuals. They may not exist for projected systems. Therefore, it may be extremely difficult to get hold of hard facts. Manufacturer contacts and advertized design rates may be the only answer. Similarly operating systems may not have been programmed and only theoretical models may exist for proposed algorithms. In this case the analyst has little choice but to make some assumptions about how the final product will turn out and proceed accordingly. The results from such simulations should be considered in light of such facts. Using Martin's guidelines the conceptual model of the operating system can be validated by the reverse reasoning process. By comparison to each previous stage, the valid conceptual model can be transformed into a valid procedural model.

When performance monitoring is the aim, validation of the job mix and hardware is greatly simplified. It is merely necessary to specify the hardware being used and to tune up the job mix to match the existing workload on the real system. The operating system is also well defined. However, it is still necessary to follow the same validation procedure to validate a model of a well defined operating system.

Provided the analyst has valid sybsystems and has correctly specified their interaction, the resulting simulation should be a close approximation to the real world. Results of the simulation can

be compared to results from other similar systems, to existing performance data or to expected values. When he is satisfied that his results are legitimate, the design and experimentation phases can begin with the simulator yielding new and important data regarding the performance of the system. Maguire [31] describes some techniques that can be used for the analysis of data obtained from experiments conducted with a simulator. In addition to the classical techniques of analysis of variance, regression, and correlation, he lists multiple ranking procedures, sequential sampling plans, spectral analysis, and Simoptimization.

The methodology developed in this thesis is valid for a wide range of computer systems. It can be used to model a very simple uniprogrammed, nonspooled environment or it can be used to model a multiprogrammed, time shared, virtual memory system. The next chapter demonstrates the diversity of the methodology by outlining two typical applications. The final chapter outlines the results and some of the limitations of the model and indicates some areas for future research.

CHAPTER 6

APPLICATION OF THE METHODOLOGY

6.1 Simulation of a Uniprogrammed System

Equipped with the methodology, it is now possible to think about simulating computer systems. In an attempt to show how the methodology can be used to simulate a hypothetical system, two examples are provided to demonstrate the application of the methodology. The first system is a uniprogrammed system. That is, only one program is allowed in main memory and it must remain there until it is completed. The second system is somewhat more complicated to show the versatility of the methodology. It is a multiprogrammed time shared system. That is, several programs are allowed in main memory at one time and jobs are not run to completion. They are only allowed to compute for a specified time slice. For simplicity's sake the objective is to compare the CPU utilization of the two systems. In a real world situation many more performance measures would be included. The description of both systems is divided into the four sections of the methodology.

The uniprogrammed system outlined above is representative of systems that existed in the early and middle sixties. Therefore it is reasonable to extract a job mix from the type of programs that were run in that era. The following form the prototype jobs for the hypothetical uniprogrammed system. Some assumption is necessary regarding the sphere in which the system operates. For example, the job mix in a university data centre would be different from that in a commercial

data centre. For this system the assumption is that it is in a university environment. A typical job mix might be:

Size	Job Type	Description	Operations
20K	1	Student - Short Compile	100 (INPUT; 10 COMPUTE) 10 (DISK INPUT; 20,000 COMPUTE; DISK OUTPUT) 100 OUTPUT
30K	2	Administration - Longer Compile	500 INPUT 50 (DISK INPUT; 10,000 COMPUTE; DISK OUTPUT) 500 (OUTPUT; 10 COMPUTE)
10K	3	Utility - Statistical Tabulation	5 (INPUT; 200 COMPUTE) (5000 COMPUTE; DISK OUTPUT) 10 (DISK OUTPUT; 100 COMPUTE; DISK OUTPUT; 10 COMPUTE) 150 COMPUTE; GET FILE 1000 (DISK INPUT; 100 COMPUTE; 3 DISK OUTPUT) 100 (DISK INPUT; 500 COMPUTE) 3000 (OUTPUT; 100 COMPUTE)
20K		Administration - Record Manipulation	(5000 COMPUTE; GET FILE; GET FILE) 100 (20 DISK INPUT; 500 COMPUTE; TAPE OUTPUT) (1000 COMPUTE; 10 DISK OUTPUT) 10 OUTPUT

Size	Job Type	Description	Operations
5K	5	Research - Compute Bound	50 (INPUT; 5 COMPUTE) 5 (DISK INPUT; 5,000,000 COMPUTE; DISK OUTPUT) 10 (OUTPUT: 10 COMPUTE)
25K	6	Research - I/O Bound	1000 (TAPE INPUT; 100 COMPUTE; TAPE OUTPUT; 10 COMPUTE)

Job arrival time must now be defined.

Job Type	Average Time Between Arrivals
1	2 minutes
2	10 minutes
3	20 minutes
4	10 minutes
5	5 minutes
6	15 minutes

The time between arrivals can be equally well described by suitable statistical distributions if required.

The hardware characteristics are specified as follows:

Device Name	Quantity	Capacity	Hardware Rates
CPU	1		50 μs Compute
Core	1	64K	22K O.S.
Disk	4	8M each	50 ms.
Tape	4	-	80 ms.
Reader	1	-	100 ms.
Printer	1	-	50 ms.
Punch	1	-	200 ms.
Terminal	1 (console)	-	500 ms.

For this simple example hardware rates are fixed and specified in terms of milliseconds (ms.). 32K of main memory is available for programs. One compute operation is estimated to require 50 microseconds.

The operating system performs as follows. When a job arrives it is assigned memory unless there is a job running, if so the job is queued. When a job receives memory it processes until completion at which point memory is released. The next job in the memory queue is assigned to memory and processing commences. The system continues until all jobs are processed. The operating system is diagrammed on the next page. The memory management scheme is an integral part of the operating system. Since the operating system employs spooling, it is possible to have a priority scheme in operation for selecting the next job to be given CPU time. In this operating system a first come--first served scheme within priority class could be used. The basic function of the input/output routine would be to interpret the type of I/O and determine the time that particular I/O operation would take from the hardware description. While file I/O is in progress the CPU remains idle.

Since the objective is to measure CPU utilization, only the time the CPU is busy need be recorded. The ratio of busy time to total elapsed time can be defined as the CPU utilization. Having defined each section of the methodology, the next step would be to convert these written descriptions to data cards or program logic, validate the model, and conduct experimental runs.

UNIPROGRAMMED OPERATING SYSTEM

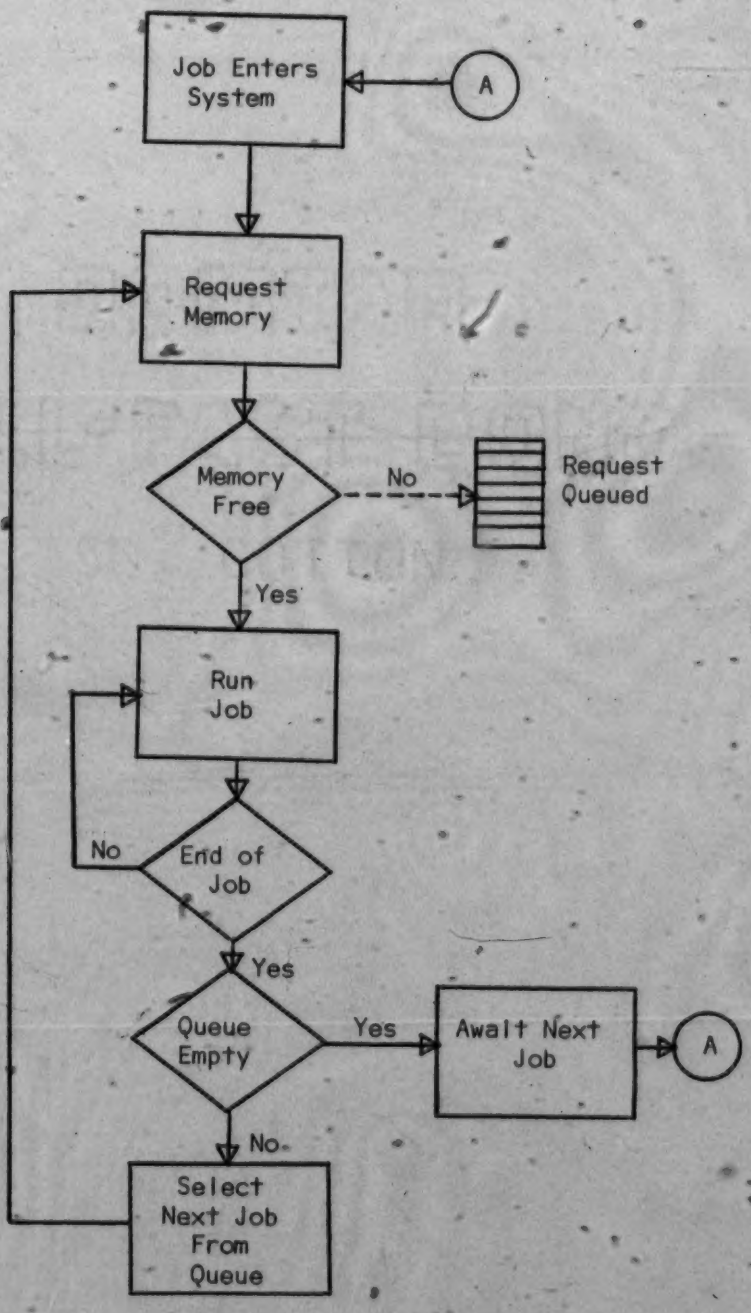


Figure 9

6.2 Simulation of a Multiprogrammed System

First of all, to properly achieve the objective (that of comparing CPU utilizations) the same job mix and hardware configuration should be used with some modifications. For example, to properly implement a multiprogramming system, more main memory is essential. However, such changes should not be detrimental to the comparison of CPU utilizations. The system performance statistics required are identical, therefore, it is only necessary to specify the operating system. Indeed the original program logic developed for the previous model can be used. It is only necessary to reprogram those sections which change as a result of a different operating system philosophy or add new sections.

The job mix remains the same. Only the changes in hardware are indicated.

Device	Quantity	Capacity	Rates
Core	1	128K	load time = 25 ms., swap time = 50 ms.
Drum	1	1M	25 ms.
Terminals	6	-	500 ms.

The operating system for this system is multiprogrammed and time shared. This means that more than one program may exist in main memory at any one time and the amount of CPU time is to be shared among all memory resident jobs. Time sharing is accomplished by switching the CPU rapidly from job to job according to some algorithm. Multiprogramming is accomplished by dividing main memory into partitions or segments and loading one program into each segment. Programs are loaded into variable sized segments unless insufficient space is available, in which case the job is placed in a memory queue. The CPU runs a

particular job until an interrupt occurs, at which time the CPU is switched to another job in memory. For purposes of this simulation the strategy used for determining which job to select next from the CPU queue is a round robin strategy. That is, each job in the CPU queue will be given one compute slice in turn until all have received time. This is not necessarily the best scheme but it does illustrate the methodology and it is simple to implement. The time slice to be used is 50 ms. If a job in the CPU queue gets CPU time but is not in memory it is swapped in from the drum device. If enough memory is not available to run the new job, jobs are swapped out until memory is available, then the new program is loaded. Processing continues until jobs complete then they are removed from the system. In theory, the system runs until all jobs are processed. In actual practice, some arbitrary elapsed time is set and the system runs until then. The flow of jobs through the operating system is illustrated on the next page. This diagram also outlines the memory management process. The I/O processor must be expanded to include Drum I/O. The same measure of CPU utilization is to be used.

The changes that would be necessary to implement the multiprogrammed system are minimal. They involve modifying the hardware description and operating system model. Within the operating system model the job control routine would have to be modified. The memory management scheme must be changed to reflect the swapping process. The scheduling routine would be altered to model the round robin strategy. The I/O processor would have to be expanded as indicated above. The hardware description is changed by modifying the input parameters.

MULTIPROGRAMMED OPERATING SYSTEM

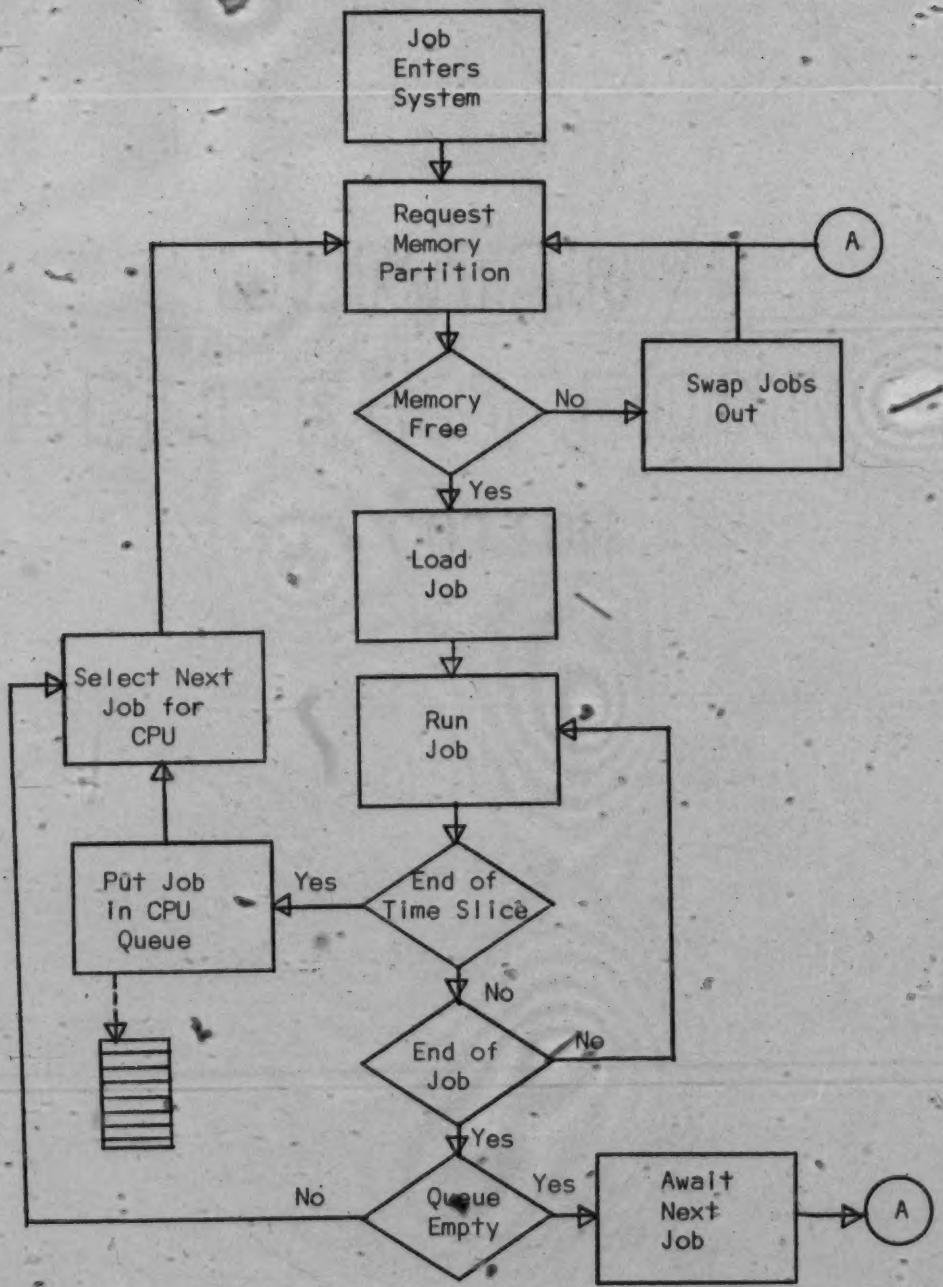


Figure 10

By suitably describing the job mix, the hardware, the operating system works, and what performance measures are required, the analyst can use the methodology described to simulate a wide range of systems.

Two applications of the methodology have been described. They show the ease with which the methodology can be used to describe a model of a computer system. In the final chapter the results of the research are stated, an indication of the limitations of the methodology is given and some areas for future research are outlined.

CHAPTER 7

CONCLUSIONS

7.1 Research Results

This investigation into the simulation of computer systems has resulted in a number of significant conclusions. From the analysis of model structure, a covering set of characteristics has been defined. This covering set consists of structural characteristics and operational characteristics. The structural characteristics include:

1. the model type
2. the method in which time advances (continuously or in discrete intervals)
3. the nature of the job mix (stochastic or deterministic)
4. the mode of operation (on-line or batch)

The operational characteristics refer to those characteristics that describe the way in which a model is implemented. These characteristics are:

1. parameterization
2. modularity
3. documentation
4. level of detail
5. system performance

Together, the structural characteristics and operational characteristics can be used to classify procedural models of computer systems.

The methodology itself evolved as a result of the study of

procedural simulators of computer systems. It describes a straightforward, flexible and cost effective method of simulating computer systems. The organization of the methodology allows the simulation of different sections at various levels of detail. The methodology is divided into four sections:

1. a job generation module
2. a hardware description
3. an operating system module
4. a system performance module

The job generator and hardware description allow a wide variety of job mixes and hardware devices to be simulated at differing levels of detail. By specifying the flow of jobs through an operating system, the input/output processing, the memory management scheme and the scheduling algorithm, a model of the operating system can be defined. By including specific measures of system performance, the construction of the model is complete.

In order to have a simulator, the model must be valid. A formal validation procedure is defined in Chapter 5. For existing systems validation can be achieved by comparison to actual data. For proposed systems, validation has to be accomplished by comparison to hypothetical or expected values. Validation is divided into two phases, validation of the conceptual model and validation of the implementation. The conceptual model can be validated by correctly specifying each module of the system, by correctly specifying intermodular activities and then, working from the conceptual model backwards, demonstrate that the original guidelines are still applicable.

The implementation is validated by checking each successive stage to the previous one.

These are the major results of the research that has been conducted. The next two sections indicate some of the limitations of the methodology and outline some of the areas for future research.

7.2 Limitations of the Methodology

Perhaps the most serious limitation of the methodology is the fact that it is not applicable to many systems at once. It is not like the generalized models (SCERT, CASE, etc.) described in Chapter 2. Only one system can be modelled at any one time. However, to alter the implementation to reflect a new operating system requires that the analyst only reprogram the event processing routines. Extensive redefinition of some data is also required in SCERT, for example, to model a given workload on a different system. The basic structure of the implementation remains the same. That is, an attempt has been made to parameterize the input and modularize the simulator for ease of modification.

Certain concepts of systems that have evolved in the last few years have not been included in the design of the methodology. For example, true parallelism is not considered as part of the model. No consideration was given to provide the ability to run virtual machines. However, the system is variable enough that modern concepts, such as those stated above, can be included if the analyst wishes to study these processes and define a method of simulating them that could be used in the context of this methodology.

No particular attention was made to the interaction among modules of the system that describe the operating system. More and

more attention is being focused on intermodular dynamics and some thought should be given to including these effects in a simulator. The subject of intermodular dynamics is discussed in more detail in the next section.

The limitations of the methodology are not serious and some concepts that have been excluded from the scope of the study due mainly to restrictions on time, can form the basis of another research project.

7.3 Future Areas of Study

Having analyzed the simulation of computer systems and having proposed a methodology to assist the analyst in his attempts to build a model, areas in which more research is needed have become apparent. This section attempts to outline some of these areas and the direction in which current efforts are moving.

Insofar as the simulation of computer systems is concerned, more research is required in the areas of representation of dynamic system modification and intermodular dynamics. Monitoring system operation to determine resource usage and, on a real time basis, to adjust the system to varying demand is a new field. More results in this area will soon appear and the specification of this type of operating system in a simulation model will have to be defined. The global performance of effects of CPU scheduling algorithms, memory management policies, arrangements of files and devices must be determined or implicitly taken into account in simulation models.

Research into the use of synthetic job streams or modules for the specification of workload is being done and should prove

fruitful. A similar concept can be applied to operating systems and synthetic operating systems can be described for use in evaluating operating systems. For real time and terminal based systems these problems are very complex. The determination of the minimal amount of detail which must be incorporated into a simulation model in order to achieve a satisfactory level of prediction is a topic of interest and one that requires additional research.

The use of on-line or interactive simulation is becoming more commonplace and needs to be studied in greater depth to ascertain its value. A good knowledge of the performance of systems under a fixed workload exists but there is little knowledge of systems under widely or rapidly varying loads.

It would be ideal to have a highly parameterized event simulator capable of simulating varying configurations on different machines. It is hoped that this thesis is a step in that direction.

BIBLIOGRAPHY

1. Aus, H.M., and Korn, G.A., "The Future On-line Continuous System Simulation", AFIPS Conf. Proceedings, Vol. 39, (1971), FJCC, pp. 379-386.

2. Bard, Y., "Performance Criteria and Measurement for a Time Sharing System", IBM Systems Journal, Vol. 10, No. 3, (1971), pp. 193-216.

3. Bard, Y., "Experimental Evaluation of System Performance", IBM Systems Journal, Vol. 12, No. 3, (1973), pp. 302-314.

4. Bell, T.E., "Objectives and Problems in Simulating Computers", AFIPS Conf. Proceedings, Vol. 41, Pt. I, (1972), FJCC, pp. 287-297.

5. Boies, S.J., "Interactive Computer User Behaviour", IBM Systems Journal, Vol. 13, No. 1, (1974), pp. 2-18.

6. Bowdon, E.K., Sr., Mamrak, S.A., and Saltz, F.R., "Simulation - A Tool for Performance Evaluation in Network Computers", AFIPS Conf. Proceedings, Vol. 42, (1973), pp. 121-131.

7. Buxton, J.N. (ed.), Simulation Programming Languages, North-Holland, 1968.

8. Cheng, P.S., "Trace-Driven System Modelling", IBM Systems Journal, Vol. 8, No. 4, (1969), pp. 280-289.

9. Chorafas, D.N., Systems and Simulation, Academic Press, 1965.

10. Cockrum, J.S., and Crockett, E.D., "Interpreting the Results of a Hardware Monitor System", AFIPS Conf. Proceedings, Vol. 38, (1971), SJCC, pp. 23-38.

11. Dahle, O.J., and Piene, J., "Evaluation of Computer Systems Through Simulation", Management Informatics, Vol. 2, No. 6, (Dec., 1973), pp. 279-283.

12. De Cegama, A., "A Methodology for Computer Model Building", AFIPS Conf. Proceedings, Vol. 40, Pt. I, (1972), FJCC, pp. 299-310.

13. Denning, P.J., "Third Generation Computing Systems", Computing Surveys, Vol. 3, No. 4, (Dec., 1971), pp. 175-216.

14. Emshoff, J.R., and Sisson, R.L., Design and Use of Computer Simulation Models, MacMillan, 1970.

15. Erickson, R.W., A Methodology for Evaluating Man-Computer Systems, M. Sc. Thesis, University of Alberta, 1971.

16. Estrin, G., Muntz, R.R., and Uzgalis, R.C., "Modelling, Measurement and Computer Power", AFIPS Conf. Proceedings, Vol. 40, (1972), SJCC, pp. 725-738.

17. Evans, G., Wallace, G.F., and Sutherland, G.L., Simulation Using Digital Computers, Prentice-Hall, 1967.

18. Fine, G.H., and McIsaac, P.V., "Simulation of a Time-Sharing System", Management Science, Vol. 12, No. 6, (Feb., 1966), pp. 180-194.

19. Gordon, G., System Simulation, Prentice-Hall, 1969.

20. Herman, D.J., "SCERT - A Computer Evaluation Tool", Datamation, Vol. 13, No. 2, (Feb., 1967), pp. 26-28.

21. Hughes, P.H., and Moe, G., "A Structural Approach to Computer Performance Analysis", AFIPS Conf. Proceedings, Vol. 42, (1973), pp. 109-120.

22. Joslin, E.O., Computer Selection, Addison-Wesley, 1968.

23. Katz, J.H., "Simulation of a Multiprocessor Computer System", AFIPS Conf. Proceedings, Vol. 28, (1966), SJCC, pp. 127-139.

24. Katz, J.H., "An Experimental Model of System/360", Comm. ACM, Vol. 10, No. 11, (Nov., 1967), pp. 684-702.

25. Katzan, H., Jr., Computer Organization and the System/370, Van-Nostrand Reinhold, 1971.

26. Kay, I.M., "An Over-the-Shoulder Look at Discrete Simulation Languages", AFIPS Conf. Proceedings, Vol. 40, (1972), SJCC, pp. 791-795.

27. Kimbleton, S.R., "The Role of Computer System Models in Performance Evaluation", Comm. ACM, Vol. 15, No. 7, (July, 1972), pp. 586-590.

28. Kochenburger, R.J., Computer Simulation in Dynamics Systems, 1972.

29. Lucas, M.C., Jr., "Performance Evaluation and Monitoring", Computing Surveys, Vol. 3, No. 4, (Sept., 1971), pp. 79-91.

30. Lynch, W.C., "Operating System Performance", Comm. ACM, Vol. 15, No. 7, (July, 1972), pp. 579-585.

31. Maguire, J.N., "Discrete Computer Simulation - Technology and Applications - The Next Ten Years", AFIPS Conf. Proceedings, Vol. 40, (1972), SJCC, pp. 815-826.

32. Maisel, H., and Gnugnoli, G., Simulation of Discrete Stochastic Systems, Science Research Associates, 1972.

33. Martin, F., Computer Modelling and Simulation, John Wiley & Sons Inc., 1968.

34. Martin, J.T., The Computerized Society, Prentice-Hall, 1970.

35. McDougall, M.H., "Simulation of an ECS-Based Operating System", AFIPS Conf. Proceedings, Vol. 30, (1967), SJCC, pp. 735-741.

36. McDougall, M.H., "Computer System Simulation: An Introduction", Computing Surveys, Vol. 2, No. 3, (Sept., 1970), pp. 191-209.

37. McKinney, J.M., "A Survey of Analytical Time-Sharing Models", Computing Surveys, Vol. 1, No. 2, (June, 1969), pp. 105-116.

38. Meier, R.C., Newell, W.T., and Pazer, H.L., Simulation in Business and Economics, Prentice-Hall, 1969.

39. Nahouraii, E., "Direct Access Device Simulation", IBM Systems Journal, Vol. 13, No. 1, (1974), pp. 19-31.

40. Naylor, T.H., "Corporate Simulation Models", Simulation, Vol. 21, No. 2, (Aug., 1973), pp. 61-64.

41. Neilsen, N.R., The Analysis of General Purpose Computer Time-Sharing Systems, Ph. D. Thesis, Stanford University, 1967.

42. Neilsen, N.R., "The Simulation of Time-Sharing Systems", Comm. ACM, Vol. 10, No. 7, (July, 1967), pp. 397-412.

43. Neilsen, N.R., "An Analysis of Some Time-Sharing Techniques", Comm. ACM, Vol. 14, No. 2, (Feb., 1971), pp. 79-90.

44. Noe, J.D., and Nutt, G.J., "Validation of a Trace-Driven CDC 6400 Simulation", AFIPS Conf. Proceedings, Vol. 40, (1972), SJCC, pp. 749-757.

45. Pinkerton, T.B., "Performance Monitoring in a Time Sharing System", Comm. ACM, Vol. 12, No. 11, (Nov., 1969), pp. 608-610.

46. Pritsker, A.A.B., and Hurst, N.R., "GASP IV: A Combined Continuous Discrete FORTRAN-Based Simulation Language", Simulation, Vol. 21, No. 3, (Sept., 1973), pp. 71-75.

47. Reitman, J., Computer Simulation Applications, John Wiley & Sons Inc., 1971.

48. Rourke, T.A., Blatney, J., and Clark, S.R., "On the Optimization of Performance of Time Sharing Systems by Simulation", Comm. ACM, Vol. 15, No. 6, (June, 1972), pp. 411-420.

49. Sammet, J., "Roster of Programming Languages for 1973", Computing Reviews, Vol. 15, No. 4, (Apr., 1974), pp. 147-160.

50. Scherr, A.L., An Analysis of Time Shared Computer Systems, Riverside Press, 1967.

51. Schmidt, J.W., Simulation and Analysis of Industrial Systems, Irwin, 1970.

52. Seaman, P.M., and Soucy, R.C., "Simulating Operating Systems", IBM Systems Journal, Vol. 8, No. 4, (1969), pp. 264-279.

53. Sherman, S., Baskett, F., and Browne, J.C., "Trace-Driven Modelling and Analysis of CPU Scheduling in a Multiprogramming System", Comm. ACM, Vol. 15, No. 12, (Dec., 1972), pp. 1063-1069.

54. Siegmann, R.M., and Gehl, J.M., "Timesharing Evaluation and Re-evaluation", Data Management, Vol. 10, No. 8, (Aug., 1972), pp. 22-25.

55. Smith, J., Computer Simulation Models, Griffin, 1968.

56. Sohnle, R.C., Tartar, J., and Sampson, J.R., "Requirements for Interactive Simulation Systems", Simulation, Vol. 20, No. 5, (May, 1973), pp. 145-152.

57. Teory, T.J., and Pinkerton, T.B., "A Comparative Analysis of Disk Scheduling Policies", Comm. ACM, Vol. 15, No. 3, (March, 1972), pp. 177-184.

58. Tocher, K.D., The Art of Simulation, English Universities Press, 1963.

59. Wagner, H.M., Principles of Management Science, Prentice-Hall, 1970.

60. Watson, R.W., Timesharing System Design Concepts, McGraw-Hill, 1970.